

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。



物联网&云平台 高级应用开发

廖建尚 编著

- ★将知识点分解到多个任务中，层次清晰、易于理解
- ★融合CC2530、传感器驱动、ZigBee、物联网、Android和云计算等技术
- ★精心设计14个趣味盎然、贴近生活的案例，加深对物联网开发的理解

物联网&云平台 高级应用开发

廖建尚 编著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书主要介绍基于 CC2530 处理器、ZigBee 无线网络的物联网和云平台开发技术,由浅入深地对物联网和云平台系统进行了介绍,采用任务式开发的学习方法,共积累了近 30 个趣味盎然、贴近生活的案例,每个案例均有完整的开发过程,都有明确的学习目标、清晰的环境开发要求、深入浅出的原理学习、详细的开发内容和完整的开发步骤。最后进行总结和拓展,将理论学习和开发实践结合起来,每个案例均附上完整的开发代码,在源代码的基础可以进行快速二次开发,读者可以快速上手。

本书涉及嵌入式系统和物联网系统的开发技术,将 CC2530 接口技术、传感器驱动、ZigBee 无线传感网络技术、物联网平台开发技术、Android 移动互联网开发结合在一起,实现了强大的物联网数据采集、传输和处理,可以开发功能强大的物联网系统,并适用在多个行业的应用。

本书可作为高等院校相关专业的教材或教学参考书,也可供相关领域的工程技术人员查阅,对于嵌入式开发、物联网系统开发和云平台开发爱好者,本书也是一本深入浅出、贴近生活的技术读物。

本书配有开发资源包,读者可登录华信教育资源网(www.hxedu.com.cn)免费注册后下载。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

物联网&云平台高级应用开发/廖建尚编著. —北京:电子工业出版社,2017.4

(物联网开发与应用丛书)

ISBN 978-7-121-31106-2

I. ①物… II. ①廖… III. ①互联网络—应用②智能技术—应用 IV. ①TP393.4②TP18

中国版本图书馆 CIP 数据核字(2017)第 055500 号

责任编辑:田宏峰

印 刷:北京京科印刷有限公司

装 订:北京京科印刷有限公司

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1 092 1/16 印张:23.25 字数:595 千字

版 次:2017 年 4 月第 1 版

印 次:2017 年 4 月第 1 次印刷

印 数:3 000 册 定价:68.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888, 88258888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式: tianhf@phei.com.cn。

FOREWORD

前言

近年来,物联网和云计算的迅猛发展,慢慢改变了社会的生产方式,大大提高了生产效率和生产力。国家规划在 9 大重点领域推广物联网,分别是智能农业、智能家居、智能电力、智能交通、智能电网、智能安防、智能物流、智能环保和智能医疗,并得到了广泛的应用且逐步改变着这些产业的结构。

物联网系统涉及的技术多、知识面广,对于一个有志于从事物联网和云平台开发的人,需要掌握微处理器的接口驱动开发技术、传感器的驱动开发技术、应用层开发技术,等等。本书介绍物联网的基本知识、开发基础,以及综合应用开发和高级应用开发,理论知识点清晰,实践案例丰富,逐步引导读者掌握物联网和云平台的开发技术并快速应用。

全书采用任务式开发的学习方法,共 14 个趣味盎然、贴近生活的案例,每个案例均有完整的开发过程,分别有明确的学习目标、清晰的环境开发要求、深入浅出的原理学习、详细的开发内容和完整的开发步骤,最后进行总结与拓展,每个案例均附上完整的开发代码,在源代码的基础可以进行快速二次开发,能方便将其转化为各种比赛的案例,或者工程技术开发人员和科研工作人员进行科研项目开发等。

第 1 章介绍了物联网基本构成和发展状况,分析了智云平台的基本框架和软硬件构成,介绍了本书开发使用的硬件平台 CC2530 的硬件资源。

第 2 章分析云平台开发技术,先介绍智云物联平台的基本使用方法,并设计了一种用于数据传输的通信协议,介绍了基于 CC2530 和 ZigBee 无线传感网络的感知层硬件开发,有采集类节点、报警类节点和控制类节点,分析了云平台的 Android 应用接口开发和 WEB 应用接口开发,并学习了云平台调试工具,最后进行了云平台的应用。

第 3 章是云平台物联网的综合应用开发,共有 7 个综合应用开发项目,分别是远程温湿度计系统、智能灯光控制系统、厨房燃气检测系统、农作物光强监测系统、GPS 网关定位系统、GSM 短信通知系统,以及视频采集与控制系统,从物联网的感知层、传输层及应用层出发,重点实现感知层和应用层的设计与开发,实现了物联网云平台的综合应用。

第 4 章是云平台物联网的高级应用开发,共有 7 个高级应用开发项目,分别是智慧窗帘控制系统、自动浇花系统、智能门禁系统、智能安防系统、实验室管理系统、无线抄表系统、智能家居自动控制系统,高级应用开发涉及感知层更多的环境信息采集和控制,从而达到物联网的智慧功能。本章也对物联网云平台知识点进行了总结,从而构建更为完整的物联网知识框架。

本书特色:

(1) 任务式开发。抛去传统的理论学习方法,选取合适的案例将理论与实践结合起来,通过理论学习和开发实践,快速入门,由浅入深掌握物联网开发技术。

(2) 各种知识点的融合。将嵌入式系统的开发技术、CC2530 处理器基本接口驱动技术、传感器驱动技术、ZigBee 无线技术、Android 移动互联网开发技术等，实现了强大的物联网数据采集、传输和处理。

本书是在另一本书籍《物联网平台开发及应用——基于 CC2530 和 ZigBee》的基础上，进一步地学习和开发综合性应用项目，建议读者先阅读该书籍的内容，夯实基础，以便快速进入本书的开发和应用。

本书既可作为高等院校相关专业师生的教学和自学参考书，也可供相关领域的工程技术人员查阅之用，对于物联网开发爱好者，本书也是一本的深入浅出的读物。

本书在编写过程中，借鉴和参考了国内外专家、学者、技术人员的相关研究成果，我们尽可能按学术规范予以说明，但难免有疏漏之处，在此谨向有关作者表示深深的敬意和谢意。如有请疏漏，请及时通过出版社与作者联系。

感谢中智讯（武汉）科技有限公司在本书编写的过程中提供的帮助，特别感谢电子工业出版社的编辑在本书出版过程中给予大力支持，该书也得到了“广东省高等职业教育品牌专业建设项目（2016gzpp044）”的资助。

由于本书涉及的知识面广，限于笔者的水平和经验，疏漏之处在所难免，恳请专家和读者批评指正。

作者
2017 年 3 月

CONTENTS 目录

第 1 章 智云物联开放平台	1
1.1 任务 1: 认识物联网	1
1.1.1 物联网	1
1.1.2 我国物联网的发展	2
1.1.3 物联网发展领域	3
1.1.4 物联网和“互联网+”	5
1.2 任务 2: 认识智云物联平台	6
1.2.1 智云物联基本框架	7
1.2.2 智云物联常用硬件	8
1.2.3 云平台可实现的项目	9
1.2.4 开发预备知识	9
1.3 任务 3: 认识物联网开发硬件	10
1.3.1 物联网开发硬件——TI CC2530 处理器	10
1.3.2 CC2530 无线节点	10
1.3.3 跳线设置及硬件连接	11
1.3.4 CC2530 无线节点硬件资源	14
1.4 任务 4: 认识 ZigBee 和 ZStack 协议栈	14
1.4.1 ZigBee 无线传感网络技术	14
1.4.2 ZStack 协议栈	15
第 2 章 云平台开发基础	21
2.1 任务 5: 智云平台配置	21
2.1.1 学习目标	21
2.1.2 开发环境	21
2.1.3 原理学习	21
2.1.4 开发内容	22
2.1.5 开发步骤	28
2.1.6 总结与拓展	32
2.2 任务 6: 认识通信协议	33
2.2.1 学习目标	33

2.2.2	开发环境	33
2.2.3	原理学习	33
2.2.4	开发内容	38
2.2.5	开发步骤	39
2.2.6	总结与拓展	41
2.3	任务 7: 硬件驱动开发	41
2.3.1	学习目标	41
2.3.2	开发环境	41
2.3.3	原理学习	41
2.3.4	开发内容	43
2.3.5	开发步骤	53
2.3.6	总结与拓展	55
2.4	任务 8: AndroidAPI 开发	55
2.4.1	学习目标	55
2.4.2	开发环境	56
2.4.3	原理学习	56
2.4.4	开发内容	61
2.4.5	开发步骤	78
2.4.6	总结与拓展	80
2.5	任务 9: WebAPI 开发	80
2.5.1	学习目标	80
2.5.2	开发环境	80
2.5.3	原理学习	80
2.5.4	开发内容	85
2.5.5	开发步骤	123
2.5.6	总结与拓展	130
2.6	任务 10: 开发调试工具	130
2.6.1	学习目标	130
2.6.2	开发环境	130
2.6.3	原理学习	130
2.6.4	开发内容	131
2.6.5	开发步骤	135
2.6.6	总结与拓展	136
2.7	任务 11: 掌握应用项目上传	136
2.7.1	学习目标	136
2.7.2	开发环境	137
2.7.3	原理学习	137
2.7.4	开发内容	137
2.7.5	开发步骤	144
2.7.6	总结与拓展	145

第3章 智云物联综合应用开发	146
3.1 任务 12: 远程温湿度计系统开发 (案例 1)	146
3.1.1 学习目标	146
3.1.2 开发环境	146
3.1.3 原理学习	146
3.1.4 开发内容	149
3.1.5 开发步骤	156
3.1.6 总结与拓展	158
3.2 任务 13: 智能灯光控制系统开发 (案例 2)	158
3.2.1 学习目标	158
3.2.2 开发环境	158
3.2.3 原理学习	158
3.2.4 开发内容	160
3.2.5 开发步骤	168
3.2.6 总结与拓展	169
3.3 任务 14: 厨房燃气检测系统开发 (案例 3)	169
3.3.1 学习目标	169
3.3.2 开发环境	169
3.3.3 原理学习	169
3.3.4 开发内容	171
3.3.5 开发步骤	180
3.3.6 总结与拓展	182
3.4 任务 15: 农作物光强监测系统开发 (案例 4)	183
3.4.1 学习目标	183
3.4.2 开发环境	183
3.4.3 原理学习	183
3.4.4 开发内容	184
3.4.5 开发步骤	192
3.4.6 总结与拓展	194
3.5 任务 16: GPS 网关定位系统开发 (案例 5)	194
3.5.1 学习目标	194
3.5.2 开发环境	194
3.5.3 原理学习	194
3.5.4 开发内容	196
3.5.5 开发步骤	200
3.5.6 总结与拓展	202
3.6 任务 17: GSM 短信通知系统开发 (案例 6)	202
3.6.1 学习目标	202
3.6.2 开发环境	202
3.6.3 原理学习	202

3.6.4	开发内容	203
3.6.5	开发步骤	205
3.6.6	总结与拓展	206
3.7	任务 18: 视频采集与控制系统开发 (案例 7)	206
3.7.1	学习目标	206
3.7.2	开发环境	206
3.7.3	原理学习	207
3.7.4	开发内容	207
3.7.5	开发步骤	216
3.7.6	总结与拓展	218
第 4 章	智云物联高级应用开发	219
4.1	任务 19: UI 设计与布局	219
4.1.1	学习目标	219
4.1.2	开发内容	219
4.1.3	开发步骤	227
4.2	任务 20: 智慧窗帘控制系统开发 (案例 8)	228
4.2.1	学习目标	228
4.2.2	开发环境	228
4.2.3	原理学习	228
4.2.4	开发内容	230
4.2.5	开发步骤	241
4.2.6	总结与拓展	242
4.3	任务 21: 自动浇花系统开发 (案例 9)	242
4.3.1	学习目标	242
4.3.2	开发环境	243
4.3.3	原理学习	243
4.3.4	开发内容	244
4.3.5	开发步骤	256
4.3.6	总结与拓展	258
4.4	任务 22: 智能门禁系统开发 (案例 10)	258
4.4.1	学习目标	258
4.4.2	开发环境	258
4.4.3	原理学习	258
4.4.4	开发内容	262
4.4.5	开发步骤	271
4.4.6	总结与拓展	273
4.5	任务 23: 智能安防系统开发 (案例 11)	274
4.5.1	学习目标	274
4.5.2	开发环境	274
4.5.3	原理学习	274

4.5.4	开发内容	277
4.5.5	开发步骤	294
4.5.6	总结与拓展	296
4.6	任务 24: 实验室管理系统开发 (案例 12)	296
4.6.1	学习目标	296
4.6.2	开发环境	296
4.6.3	原理学习	296
4.6.4	开发内容	298
4.6.5	开发步骤	315
4.6.6	总结与拓展	317
4.7	任务 25: 无线抄表系统开发 (案例 13)	317
4.7.1	学习目标	317
4.7.2	开发环境	317
4.7.3	原理学习	317
4.7.4	开发内容	319
4.7.5	开发步骤	334
4.7.6	总结与拓展	335
4.8	任务 26: 智能家居自动控制系统开发 (案例 14)	336
4.8.1	学习目标	336
4.8.2	开发环境	336
4.8.3	原理学习	336
4.8.4	开发内容	337
4.8.5	开发步骤	344
4.8.6	总结与拓展	347
附录 A	常见硬件及问题	348
A.1	Android 智云 Android 开发平台的使用	348
A.2	无线节点镜像固化	350
A.3	无线节点修改网络信息	351
A.4	无线节点读取 IEEE 地址	352
A.5	认识各种传感器	353
A.6	传感器操作说明	357
参考文献		360

物联网

物联网的含义是：第一，物联网的核心和基础仍然是互联网，是在互联网基础上延伸和扩展的网络；第二，其用户端延伸和扩展到了任何物品，以及物品之间进行信息交换和通信。因此，物联网是指运用传感器、射频识别 (RFID)、智能嵌入式等技术，使信息传感设备

智云物联开放平台

本章引导读者初步认识物联网和云平台，并介绍智云平台和物联网开发硬件，让开发者初步了解物联网开发过程。

1.1 任务 1：认识物联网

1.1.1 物联网

物联网（Internet of Things）的概念最早于 1999 年，由美国麻省理工学院首次提出。2009 年初 IBM 抛出了“智慧地球”概念，使得物联网成为时下热门话题。2009 年 8 月，温总理提出启动“感知中国”建设，随后物联网在中国进一步升温，得到政府、科研院校、电信运营商及设备提供商等相关厂商的高度重视。

物联网（Internet of Things），是指利用各种信息传感设备，如射频识别（RFID）装置、无线传感器、红外感应器、全球定位系统、激光扫描器等对现有物品信息进行感知、采集，通过网络支撑下的可靠传输技术，将各种物品的信息汇入互联网，并进行基于海量信息资源的智能决策、安全保障及管理技术与服务的全球公共的信息综合服务平台，如图 1.1 所示。

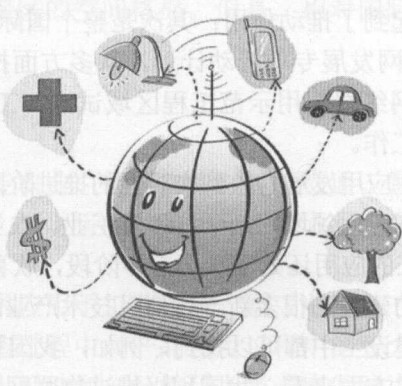
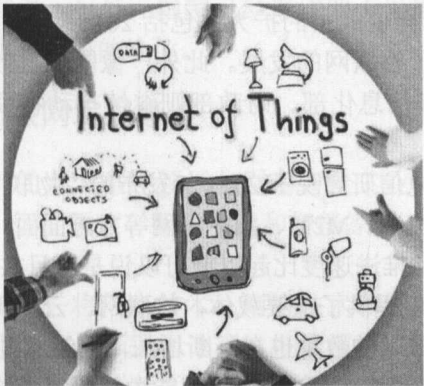


图 1.1 物联网

物联网有两层意思：第一，物联网的核心和基础仍然是互联网，是在互联网基础上延伸和扩展的网络；第二，其用户端延伸和扩展到了任何物品，以及物品之间进行信息交换和通信。因此，物联网是指运用传感器、射频识别（RFID）、智能嵌入式等技术，使信息传感设备



感知任何需要的信息，按照约定的协议，通过网络（如基于 Wi-Fi 的无线局域网、3G/4G 等）接入方式，把任何物体与互联网相连接，进行信息交换通信，在进行物与物、物与人的泛在连接的基础上，实现对物体的智能化识别、定位、跟踪、控制和管理。物联网架构，分为感知识别层、网络传输层、综合应用层，如图 1.2 所示。



图 1.2 物联网架构示意图

1.1.2 我国物联网的发展

不断增长的物联网行业如图 1.3 所示。

首先，政策日趋完善。我国物联网从 2009 年发展以来，从 2012 年开始加大顶层设计的力度，2012 年 8 月份，以物联网专家委员会成立为标志，这两年政府层面推动了很多顶层设计的工作：首先是以物联网指导意见为标志的 2013 年 2 月份国务院 7 号文发布，对整个物联网的发展起到了推动作用；其次是整个国际联席会议成员的扩大，包括 2015 年 9 月份印发的 10 个物联网发展专项行动计划，从多方面推动了物联网的发展。此外，像国家发改委开展的国家物联网终端应用示范工程区域试点，工业和信息化部、财政部则继续推动物联网发展专项资金的工作。

另外，应用发展进入到实质性的推进阶段。电信研究院于 2014 年发布的“物联网白皮书”列出了很多应用领域的例子，涉及工业、农业、交通、M2M、智能电网等方方面面。但同时看到，现在的应用还处于一个起步阶段，欣喜的是推进速度比起以前可以说是有目共睹。同时，智慧城市的建设为很多新一代信息技术产业的应用提供了重要载体，物联网、云计算、大数据的应用在建设当中都可以找到，例如，我国智慧城市的数量也在不断增长，已经超过 300 个。

从技术方面来看，我国积极推进物联网自主技术标准和共性基础能力的研究，物联网架构对整个物联网发展非常重要，国内也一直试图在物联网架构设计上能有国内自主创新的东西。

在架构研究上业界达成了统一的共识，就是物联网的发展要借鉴互联网开放的理念，包括它的运营的体系系统及 IP 系统，所以也从可扩展性、泛技术性、服务保障性等方面进行了需求的归纳。我国的技术创新主要体现在一些传感器技术上的突破，包括 RFID 上的创新，以及面向工业控制的 WIA-PA 标准。我国是 ITU 和 ISO 对应工作组的主导国之一，在 M2M 国



际标准化组织中也有很多处于领导地位。

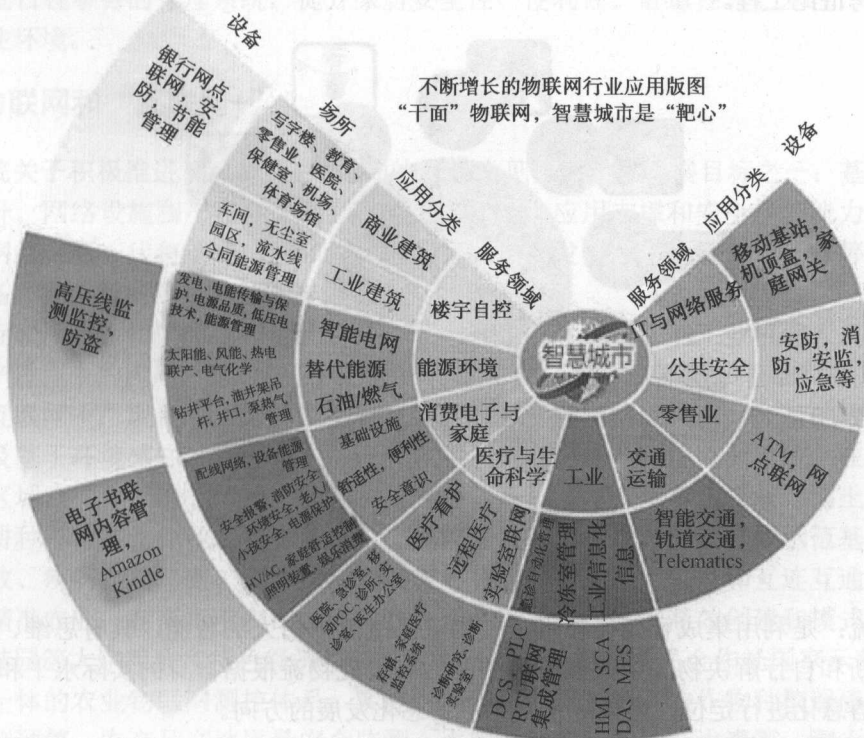


图 1.3 不断增长的物联网行业

从产业方面来看,我国物联网产业体系相对完善,局部领域获得突破,整体领域保持较快增长。2013年年底,我国物联网产业规模达到5000亿,在制造这个环节,获得了局部的突破,如RFID技术,以及工业芯片等方面都取得较大的突破。在物联网服务方面,M2M是整个产业的亮点。另外,国家已经形成四大发展集聚区的空间格局。但是,相对国际来讲,还仍然处于非常弱势的地位。

1.1.3 物联网发展领域

物联网的九大重点领域分别为智能工业、智能农业、智能物流、智能交通、智能环保、智能安防、智能医疗、智能物流和智能家居，如图 1.4 所示，物联网已经深入社会生活的方方面面。

智能工业：将信息技术、网络技术和智能技术应用于工业领域、给工业注入“智慧”的综合技术。它突出了采用计算机技术模拟人在制造过程中和产品使用过程中的智力活动，以进行分析、推理、判断、构思和决策，从而去扩大延伸和部分替代人类专家的脑力劳动，实现知识密集型生产和决策自动化。

智能农业：在相对可控的环境条件下，采用工业化生产，实现集约高效可持续发展的现代超前农业生产方式，就是农业先进设施与陆地相配套、具有高度的技术规范和高效益的集约化规模经营的生产方式。它集科研、生产、加工、销售于一体，实现周年性、全天候、反季节的企业化规模生产；它集成现代生物技术、农业工程、农用新材料等学科，以现代化农



业设施为依托，科技含量高，产品附加值高，土地产出率高和劳动生产率高，是我国农业新技术革命的跨世纪工程。

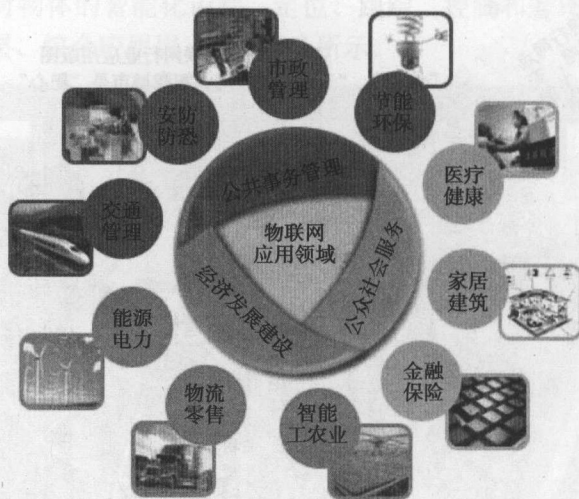


图 1.4 物联网九大重点领域

智能物流：是利用集成智能化技术，使物流系统能模仿人的智能，具有思维、感知、学习、推理判断和自行解决物流中某些问题的能力。智能物流根据自身的实际水平和客户需求对智能物流信息化进行定位，是国际未来物流信息化发展的方向。

智能交通：智能交通系统（ITS）是未来交通系统的发展方向，它是将先进的信息技术、数据通信传输技术、电子传感技术、控制技术及计算机技术等有效地集成运用于整个地面交通管理系统而建立的一种在大范围内、全方位发挥作用的，实时、准确、高效的综合交通运输管理系统。

智能电网：电网的智能化，也被称为“电网 2.0”，它是建立在集成的、高速双向通信网络的基础上，通过先进的传感和测量技术、先进的设备技术、先进的控制方法以及先进的决策支持系统技术的应用，实现电网的可靠、安全、经济、高效、环境友好和使用安全的目标，其主要特征包括自愈、激励和包括用户、抵御攻击、提供满足 21 世纪用户需求的电能质量、容许各种不同发电形式的接入、启动电力市场以及资产的优化高效运行。

智能环保：在原有“数字环保”的基础上，借助物联网技术，把感应器和装备嵌入到各种环境监控对象（物体）中，通过超级计算机和云计算将环保领域物联网整合起来，实现人类社会与环境业务系统的整合，以更加精细和动态的方式实现环境管理和决策的“智慧”。“智慧环保”是“数字环保”概念的延伸和拓展，是信息技术进步的必然趋势。

智能安防：通过相关内容的服务的信息化、图像的传输和存储、数据的存储和处理等，实现企业或住宅、社会治安、基础设施及重要目标的智能化安全防范。

智能医疗：通过打造健康档案区域医疗信息平台，利用最先进的物联网技术，实现患者与医务人员、医疗机构、医疗设备之间的互动，逐步达到信息化。在不久的将来医疗行业将融入更多人工智慧、传感技术等高科技，使医疗服务走向真正意义的智能化，推动医疗事业的繁荣发展。在中国新医改的大背景下，智能医疗正在走进寻常百姓的生活。

智能家居：以住宅为平台，利用综合布线技术、网络通信技术，智能家居系统设计方案



将安全防范技术、自动控制技术、音视频技术将家居生活有关的设施集成,构建高效的住宅设施与家庭日程事务的管理系统,提升家居安全性、便利性、舒适性、艺术性,并实现环保节能的居住环境。

1.1.4 物联网和“互联网+”

国务院关于积极推进“互联网+”行动的指导意见明显提出发展目标之一:基础支撑进一步夯实提升。网络设施和产业基础得到有效巩固加强,应用支撑和安全保障能力明显增强。固定宽带网络、新一代移动通信网和下一代互联网加快发展,物联网、云计算等新型基础设施更加完备。人工智能等技术及其产业化能力显著增强。

在“互联网+”协同制造方面加快推动云计算、物联网、智能工业机器人、增材制造等技术在生产过程中的应用,推进生产装备智能化升级、工艺流程改造和基础数据共享。

在“互联网+”现代农业推广成熟可复制的农业物联网应用模式。在基础较好的领域和地区,普及基于环境感知、实时监测、自动控制的网络化农业环境监测系统。在大宗农产品规模生产区域,构建天地一体的农业物联网测控体系,实施智能节水灌溉、测土配方施肥、农机定位耕种等精准化作业。在畜禽标准化规模养殖基地和水产健康养殖示范基地,推动饲料精准投放、疾病自动诊断、废弃物自动回收等智能设备的应用普及和互连互通。引导各地大力发展精准农业,在高标准农田、现代农业示范区、绿色高产高效创建和模式攻关区、园艺作物标准园等大宗粮食和特色经济作物规模生产区域,以及农民合作社国家示范社等主体,构建天地一体的农业物联网测控体系,实施农情信息监测预警、农作物种植遥感监测、农作物病虫监测预警、农产品产地质量安全监测、水肥一体化和智能节水灌溉、测土配方施肥、农机定位耕种等精准化作业。大力推进物联网在农业生产中的应用,在国家现代农业示范区率先取得突破;建成一批大田种植、设施园艺、畜禽养殖、水产养殖物联网示范基地;研发一批农业物联网产品和技术,熟化一批农业物联网成套设备,推广一批节本增效农业物联网应用模式,加强推广应用。重点加强成熟度、营养组分、形态、有害物残留、产品包装标识等传感器研发,推进动植物环境(土壤、水、大气)、生命信息(生长、发育、营养、病变、胁迫等)传感器熟化,促进数据传输、数据处理、智能控制、信息服务的设备和软件开发。研究物联网技术在不同产品、不同领域的集成、组装模式和技术实现路径,促进农业物联网基础理论研究,探索构建国家农业物联网标准体系及相关公共服务平台。推进农业生产集约化、工程装备化、作业精准化和信息化管理,为农业物联网广泛应用奠定基础。

“互联网+”高效物流。探索能源消费新模式,开展绿色电力交易服务区域试点,推进以智能电网为配送平台,以电子商务为交易平台,融合储能设施、物联网、智能用电设施等硬件以及碳交易、互联网金融等衍生服务于一体的绿色能源网络发展,实现绿色电力的点到点交易及实时配送和补贴结算。进一步加强能源生产和消费协调匹配,推进电动汽车、港口岸电等电能替代技术的应用,推广电力需求侧管理,提高能源利用效率。基于分布式能源网络,发展用户端智能化用能、能源共享经济和能源自由交易,促进能源消费生态体系建设。

“互联网+”智慧能源。建设深度感知智能仓储系统。在各级仓储单元积极推广应用二维码、无线射频识别等物联网感知技术和大数据技术,实现仓储设施与货物的实时跟踪、网络化管理以及库存信息的高度共享,提高货物调度效率。鼓励应用智能化物流装备提升仓储、运输、分拣、包装等作业效率,提高各类复杂订单的出货处理能力,缓解货物囤积停滞瓶颈。



制约，提升仓储运管水平和效率。

“互联网+”便捷交通。推进交通运输资源在线集成。利用物联网、移动互联网等技术，进一步加强对公路、铁路、民航、港口等交通运输网络关键设施运行状态与通行信息的采集。推动跨地域、跨类型交通运输信息互联互通，推广船联网、车联网等智能化技术应用，形成更加完善的交通运输感知体系，提高基础设施、运输工具、运行信息等要素资源的在线化水平，全面支撑故障预警、运行维护以及调度智能化。

“互联网+”绿色生态。完善废旧资源回收利用体系。利用物联网、大数据开展信息采集、数据分析、流向监测，优化逆向物流网点布局。支持利用电子标签、二维码等物联网技术跟踪电子废物流向，鼓励互联网企业参与搭建城市废弃物回收平台，创新再生资源回收模式。加快推进汽车保险信息系统、“以旧换再”管理系统和报废车管理系统的标准化、规范化和互联互通，加强废旧汽车及零部件的回收利用信息管理，为互联网企业开展业务创新和便民服务提供数据支撑。

1.2 任务 2：认识智云物联平台

一个物联网应用，一般要完成传感器数据的采集、存储以及数据的加工和处理这三项工作。举例来说，对于驾驶员，希望获取去目的地的路途上的路况，为了完成这个目标，就需要大量的交通流量传感器对几个可能路线上的车流和天气状况进行实时的采集，并存储到集中的路况处理服务器，应用在服务器上通过适当的算法，从而得出大概的到达时间，并将处理的结果展示给驾驶员。所以系统架构设计可以分为如下三部分。

- 传感器硬件和接入互联网的通信网关（负责将传感器数据采集起来，发送到互联网服务器）；
- 高性能的数据接入服务器和海量存储；
- 特定应用，处理结果展现服务。

要实现物联网系统架构，需要一个基于云计算与互联网的平台加以支撑，而这个平台的稳定性、可靠性、易用性，对物联网项目顺利实施起着非常关键的作用。智云物联公共服务平台就是这样的一个开放平台，实现了物联网服务平台的主要基础功能开发，提供开放程序接口，为开发者提供基于互联网的物联网应用服务。

智云物联是一个开放的公共物联网接入平台，目的是服务所有物联网爱好者和开发者，使物联网传感器数据的接入、存储和展现变得轻松简单，让开发者能够快速开发出专业的物联网应用系统，如图 1.5 所示。

其中智云物联平台具有以下特点。

- 让无线传感网快速接入到互联网和电信网，支持手机和 Web 远程访问及控制；
- 解决多用户对单一设备访问的互斥，数据对多用户的主动消息推送等技术难题；
- 开源稳定的底层工业级传感网络协议栈-ZigBee 协议；
- 开源的传感器硬件驱动库，开源的应用项目资源；
- 免应用编程的 BS 项目发布系统、Android 组态系统接入系统。



图 1.5 智云物联平台

1.2.1 智云物联基本框架

智云物联公共服务平台在移动互联/物联网项目架构中框架如图 1.6 所示。



图 1.6 智云平台框架

(1) 全面感知。提供 CC2530 为核心板的环境感知数据采集，如温度、湿度、光照度等常见的环境感知，有成熟的传感器接口开发包，适合二次开发和项目深研。

(2) 网络传输。实现 ZigBee、Wi-Fi、Bluetooth 等无线/有线通信技术；采用易懂易学的 JSON 数据通信格式的 ZXBee 轻量级通信协议；多种智能 M2M 网关，如 ZCloud-GW-S4418、



ZCloud-GW-9x25、ZCloud-GW-PC，集成 Wi-Fi/3G/100M 以太网等网络接口，支持本地数据推送及远程数据中心接入，采用 AES 加密认证。

(3) 数据中心。高性能工业级物联网数据集群服务器，支持海量物联网数据的接入、分类存储、数据决策、数据分析及数据挖掘；分布式大数据技术，具备数据的即时消息推送处理、数据仓库存储与数据挖掘等功能；云存储采用多处备份，数据永久保存，数据丢失概率小于 0.1%；基于 B/S 架构的后台分析管理系统，支持 Web 对数据中心进行管理和系统运营监控。

主要功能模块有消息推送、数据存储、数据分析、触发逻辑、应用数据、位置服务、短信通知、视频传输等。

(4) 应用服务。智云物联开放平台应用程序编程接口，提供 SensorHAL 层、Android 库、Web JavaScript 库等 API 二次开发编程接口，具有互联网/物联网应用所需的采集、控制、传输、显示、数据库访问、数据分析、自动辅助决策、手机/Web 应用等功能，可以基于该 API 上开发一整套完整的互联网/物联网应用系统。

提供实时数据（即时消息）、历史数据（表格/曲线）、视频监控（可操作云台转动、抓拍、录像等）、自动控制、短信/GPS 等编程接口。

提供 Android 和 Windows 平台下数据分析测试工具，方便程序的调试及测试；基于开源的 JSP 框架的 B/S 应用服务，支持用户注册及管理、后台登录管理等基本功能，支持项目属性和前端页面的修改，能够根据项目需求定制各个行业应用服务，例如，智能家居管理平台、智能农业管理平台、智能家庭用电管理平台、工业自动化专家系统……

Android 应用组态软件，支持各种自定义设备，包括传感器、执行器、摄像头等的动态添加、删除和管理，无须编程即可完成不同应用项目的构建。

1.2.2 智云物联常用硬件

智云物联平台支持各种智能设备的接入，硬件模型如图 1.7 所示。



图 1.7 硬件模型

(1) 传感器：主要用于采集物理世界中发生的物理事件和数据，包括各类物理量、标识、音频、视频数据。

(2) 节点：采用 CC2530 和 ARM 等微控制器，具备物联网传感器的数据的采集、传输、组网能力，能够构建传感网络。

(3) 网关：实现传感网与电信网/互联网的数据连通，支持 ZigBee、Wi-Fi、蓝牙等多种传感协议的数据解析，支持网络路由转发，实现 M2M 数据交互。

(4) 云服务器：负责对物联网海量数据进行中央处理，运行云计算大数据技术实现对数据的存储、分析、计算、挖掘和推送功能，并采用统一的开放接口为上层应用提供数据服务。

(5) 应用终端：运行物联网应用的移动终端，如 Android 手机、平板等设备。



1.2.3 云平台可实现的项目

采用智云物联开放平台框架，完成各种物联网应用项目开发，实现多种应用，详细介绍参考网页介绍（<http://www.zhiyun360.com/docs/01xsrm/03.html>），如图 1.8 所示。

			
智能家居	智能农业	远程抄表	智能仓储
			
智能医疗	水产养殖	智能工厂	仪器预约
			
智能电网	智能交通	智能电梯	食品溯源
			
家居能耗	雾霾监测	智能小车	无线考勤

图 1.8 能实现的项目

1.2.4 开发预备知识

本节主要指引开发者快速学习基于智云物联公共服务平台快速开发移动互联、物联网的综合项目，学习智云物联产品前，要求开发者预先学习以下基本知识和技能。

- 了解和掌握基于 CC2530 的单片机接口技术/传感器接口技术；
- 了解 ZigBee 无线传感网基础知识，及基于 CC2530 的 ZigBee ZStack 组网原理；
- 了解和掌握 Java 编程，掌握 Android 应用程序开发；
- 了解和掌握 HTML、JavaScript、CSS、Ajax 开发，熟练使用 DIV+CSS 进行网页设计；
- 了解和掌握 JDK+ApacheTomcat+Eclipse 环境搭建及网站开发。



1.3 任务 3：认识物联网开发硬件

1.3.1 物联网开发硬件——TI CC2530 处理器

ZigBee 新一代 SoC 芯片 CC2530 是真正的片上系统解决方案，支持 IEEE 802.15.4 标准、ZigBee、ZigBee RF4CE 和能源的应用。CC2530 拥有 256 B 的快闪记忆体，是理想 ZigBee 应用芯片，支持 RemoTI 的 ZigBee RF4CE，它是业界首款符合 ZigBee RF4CE 协议栈的芯片，更大内存将允许芯片无线下载、支持系统编程。此外，CC2530 结合了一个完全集成的、高性能的 RF 收发器与一个 8051 微处理器、8 KB 的 RAM、32/64/128/256 KB 闪存，以及其他强大的功能和外设，拥有以下特性。

(1) 强大无线前端。2.4 GHz 的 IEEE 802.15.4 标准射频收发器，出色的接收器灵敏度和抗干扰能力，可编程输出功率为+4.5 dBm，总体无线连接 102 dBm。

(2) 低功耗。接收模式为 24 mA，发送模式（1 dBm）为 29 mA，功耗模式 1（4 μ s 唤醒）为 0.2 mA，功率模式 2（睡眠计时器运行）为 1 μ A，功耗模式 3（外部中断）为 0.4 μ A，宽电源电压范围为 2~3.6 V。

(3) 微控制器。高性能和低功耗 8051 微控制器内核，32/64/128/256 KB 系统可编程闪存，8 KB 的内存保持在所有功率模式，支持硬件调试。

(4) 丰富的外设接口。强大的 5 通道 DMA，IEEE 802.15.4 标准的 MAC 定时器，通用定时器（1 个 16 位、2 个 8 位），红外发生电路，32 kHz 的睡眠计时器和定时捕获，精确的数字接收信号强度指示支持，电池监视器和温度传感器，8 通道 12 位 ADC，可配置分辨率，2 个强大的通用同步串口，21 个通用 I/O 引脚，看门狗定时器。

(5) 可应用领域。2.4 GHz 的 IEEE 802.15.4 标准系统、ZigBee 系统、楼宇自动化、照明系统、工业控制和监测、低功率无线传感器网络、消费电子、健康照顾和医疗保健。

1.3.2 CC2530 无线节点

无线节点一般安装在任务箱内或者独立使用，主要由嵌入式底板、无线模组、传感器、LCD 屏四部分组成，普通型节点不含 LCD 屏，且嵌入式底板不包含 ARM 芯片。

无线节点硬件如图 1.9 所示。

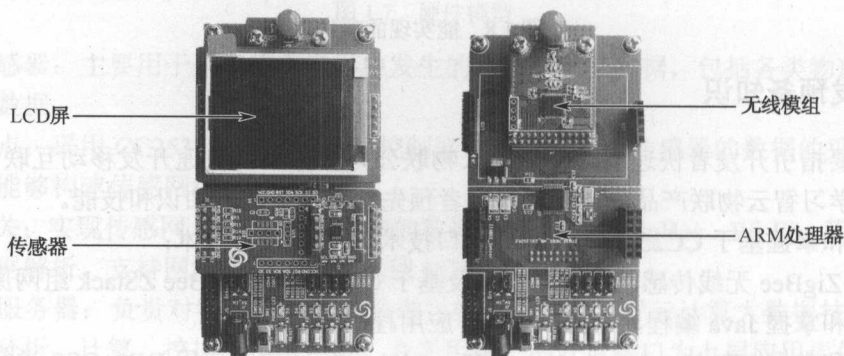


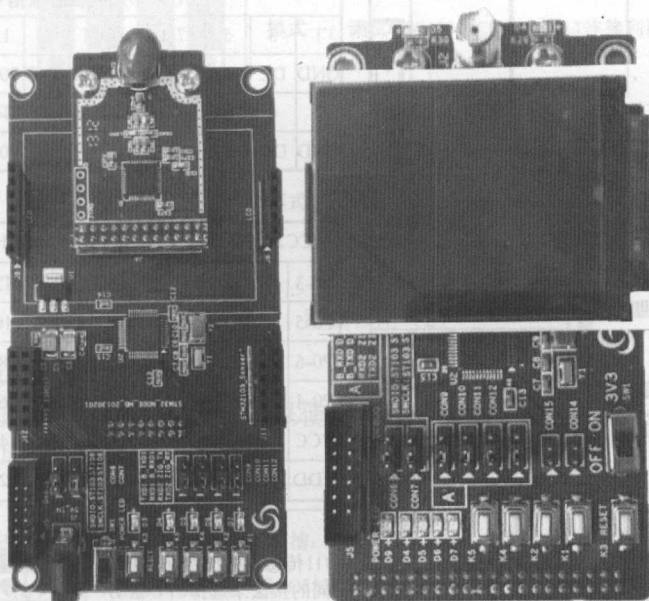
图 1.9 无线节点硬件



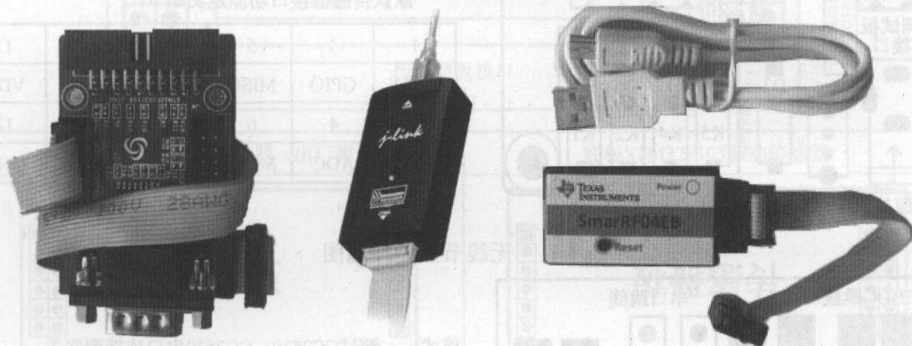
1.3.3 跳线设置及硬件连接

1. ZXBeeEdu 硬件

ZXBeeEdu (+) 系列无线节点及配件实物如图 1.10 所示。



(a) 无线节点无线协调器



(b) 无线节点调试接口板 ARM Cortex 仿真器 CC2530 仿真器

图 1.10 ZXBee 几个主要模块实物图

2. 硬件结构框图

硬件结构如图 1.11 所示。

3. 跳线说明

(1) 无线协调器跳线说明：无线协调器直接安装到任务主板对应插槽中，跳线使用如图 1.12 所示。

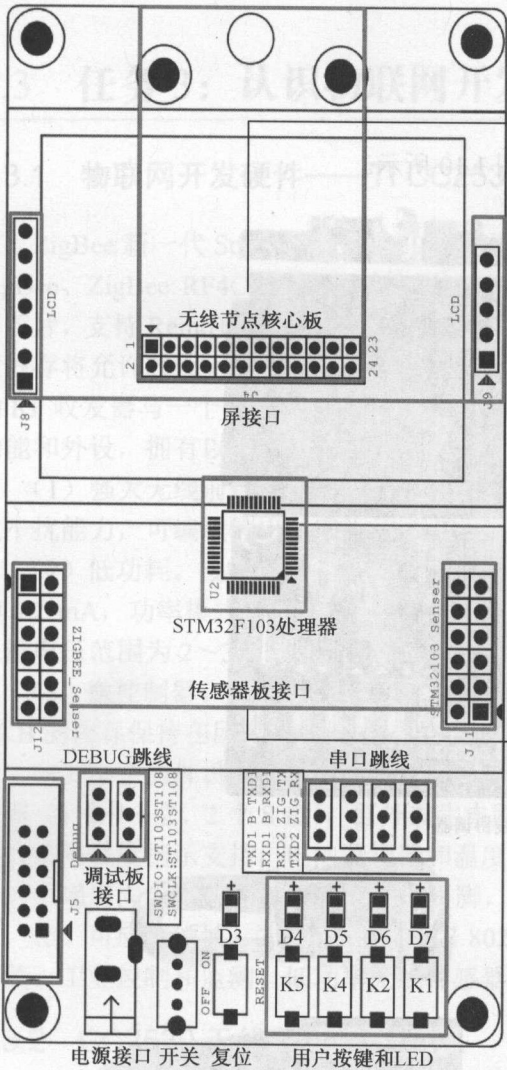






图 1.11 无线节点硬件框图

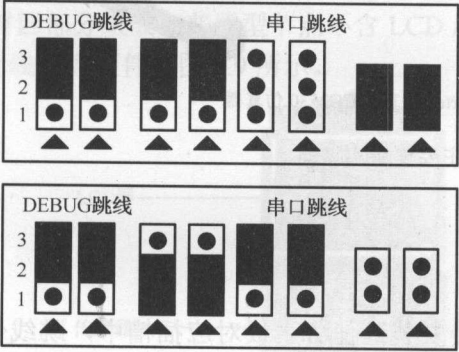
CC2530无线核心板信号定义																							
1	3	5	7	9	11	13	15	17	19	21	23												
GND	DC	P2-0	P1-6	P1-4	P1-2	P1-0	P0-0	P0-2	P0-4	P0-6	VCC												
2	4	6	8	10	12	14	16	18	20	22	24												
GND	DD	P1-7	P1-5	P1-3	P1-1	RST	P0-1	P0-3	P0-5	P0-7	VCC												

CC2530驱动传感器接口										STM32F103驱动传感器接口									
P0-3	1			2	P0-2	GND	12			11	VDD								
P0-5	3			4	P0-1	PB1	10			9	VCC								
P0-6	5			6	P1-3	PA5	8			7	PA4								
P0-4	7			8	P1-0	PA7	6			5	PA6								
VCC	9			10	P1-1	PB0	4			3	PB5								
VDD	11			12	GND	PB11	2			1	PB10								

左边的J12传感器接口接CC2530无线核心板；右边的J11传感器接口接底板STM32F103处理器。通过传感器板不同的接法来选择CPU驱动。此处VDD为5.0 V，VCC为3.3 V

默认传感器接口功能定义如下：

1	3	5	7	9	11
TXD	GPIO	MISO	CS	VCC	VDD
2	4	6	8	10	12
RXD	ADC	MOSI	SCK	PWM	GND



模式一：调试CC2530，CC2530串口连接到网关
(运行ZigBee ZStack协议栈默认设置)

模式二：调试CC2530，CC2530串口连接到调试扩展板

图 1.12 跳线使用（1）

(2) 无线节点跳线说明：ZXBee 系列无线节点板上提供了两组跳线用于选择调试不同处理器，跳线使用如图 1.13 所示。



4. 传感器板的使用

传感器板可以有两种接法，分别通过无线核心板（CC2530）和底板 STM32F103 驱动，如图 1.13 所示。

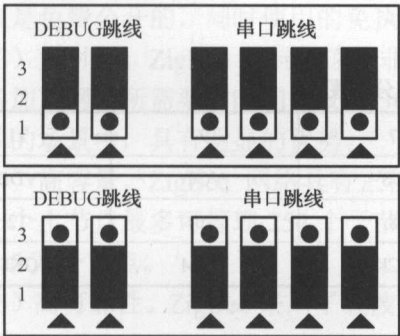


图 1.13 跳线使用（2）

5. 调试接口板的使用

通过调试接口板的转接，无线节点可以使用仿真器进行调试，同时还可以使用 RS232 串口，连接如图 1.14 所示。

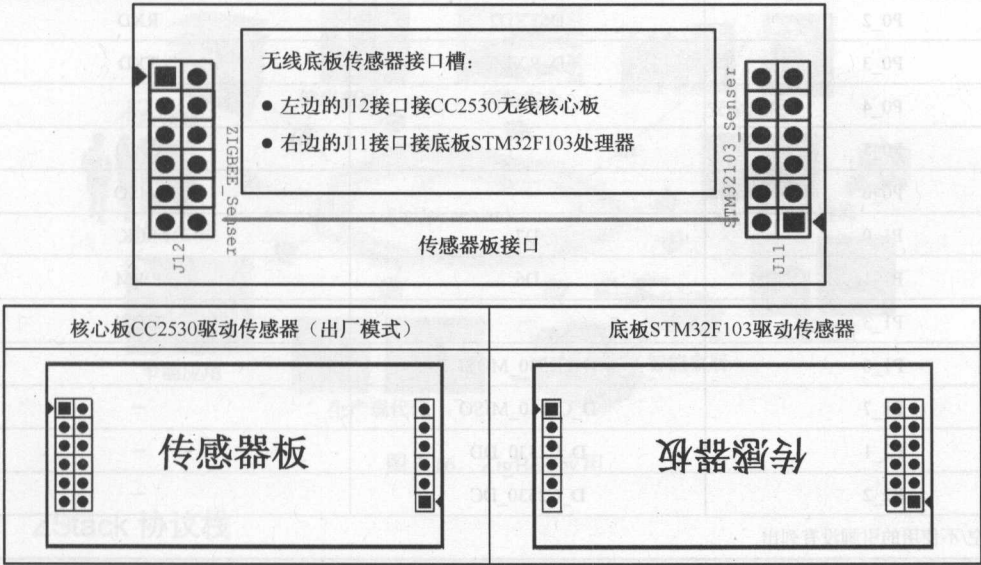


图 1.14 传感器板接法

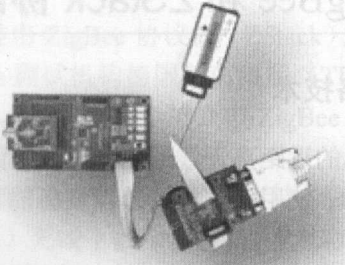


图 1.15 调试接口板的连接



1.3.4 CC2530 无线节点硬件资源

1. 传感器接口引脚

传感器接口引脚分配见表 1.1。

表 1.1 传感器接口引脚分配图

1	3	5	7	9	11
TXD	GPIO	MISO	CS	VCC	VDD
2	4	6	8	10	12
RXD	ADC	MOSI	SCK	PWM	GND

2. CC2530 无线节点硬件资源分配

CC2530 无线节点硬件资源分配见表 1.2。

表 1.2 CC2530 无线节点硬件资源

引 脚	底 板 设 备	传感器接口
P0_1	K4	ADC
P0_2	D_TXD2	RXD
P0_3	D_RXD2	TXD
P0_4	K5	CS
P0_5	—	GPIO
P0_6	—	MISO
P1_0	D7	SCK
P1_1	D6	PWM
P1_3	—	MOSI
P1_6	D_C2530_MOSI	—
P1_7	D_C2530_MISO	—
P2_1	D_C2530_DD	—
P2_2	D_C2530_DC	—

注：悬空/不使用的引脚没有列出

1.4 任务 4：认识 ZigBee 和 ZStack 协议栈

1.4.1 ZigBee 无线传感网络技术

ZigBee 技术是近些年才兴起的近距离无线通信技术，是无线传感器网络（Wireless Sensor Network, WSN）的核心技术之一，使用该技术的节点设备能耗特别低、自组网无须人工干预、成本低廉、设备复杂度低且网络容量大，特点如下。

（1）低功耗。这是 ZigBee 最具代表性的特点。该技术具有低速率，以及低发射功率的特



性,另外休眠功能使其设备的功耗进一步降低。两节普通5号干电池便可使其设备正常工作6~24个月,而使用其他技术的设备根本无法做到这一点。

(2) 低成本。ZigBee 协议具有简单明了的特点,对相关设备要求也不是很高,除此之外,该协议是免费公开的,同时使用的免执照频段,也缩减了其使用成本。

(3) 短时延。ZigBee 的响应速度非常快,当有事件触发时,只需要 15 ms 的反应时间,而设备加入网络所需要的时间也仅有 30 ms,同时功耗也得到降低。因此该技术在定时延有较高要求的场景中,具有明显的优势。

(4) 高容量。ZigBee 网络具有三种拓扑结构,由一个中心节点对整个网络进行管理与维护。一个主节点最多可管理 254 个子节点,再加上灵活的组网方式,一个 ZigBee 网络最多可包含 6.5 万个节点。

(5) 高可靠性。ZigBee 采用了载波侦听多点接入/冲突避免的机制,即 CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) 来保证通信的高可靠性,同时通过预留专用时隙的方式避免数据传送过程中的竞争与冲突。

ZigBee 技术本身,针对低数据量、低成本、低功耗、高可靠性的无线数据通信的需求而产生的。在多方面领域有广泛应用,在国防安全、工业应用、交通物流、节能、生产现代化和智能家居等领域有着广泛应用。



图 1.16 ZigBee 应用

1.4.2 ZStack 协议栈

CC2530 芯片支持 ZigBee 协议,同时 TI 公司提供的 ZStack 协议栈则是一套符合 ZigBee 协议的协议栈。

2007 年 1 月, TI 公司宣布推出 ZigBee 协议栈 (ZStack), 并于 2007 年 4 月提供免费下载版本 V1.4.1。ZStack 达到 ZigBee 测试机构德国莱茵集团 (TUV Rheinland) 评定的 ZigBee 联盟参考平台 (Golden Unit) 水平, 目前已为全球众多 ZigBee 开发商所广泛采用。ZStack 符合 ZigBee 2006 规范, 支持多种平台。

除了全面符合 ZigBee 2006 规范以外, ZStack 还支持丰富的新特性, 如无线下载, 可通过 ZigBee 网状网络 (Mesh Network) 无线下载节点更新。ZStack 还支持具备定位感知 (Location Awareness) 特性的 CC2431。上述特性使开发者能够设计出可根据节点当前位置改变行为的新



型 ZigBee 应用。

2007 年 7 月，ZStack 升级为 V1.4.2，之后对其进行了多次更新，并于 2008 年 1 月升级为 V1.4.3。2008 年 4 月，针对 MSP430F4618+CC2420 组合把 ZStack 升级为 V2.0.0，2008 年 7 月，ZStack 升级为 V2.1.0，2009 年 4 月，Z-Stack 支持符合 2.4 GHz IEEE 802.15.4 标准的第二代片上系统 CC2530；2009 年 9 月，ZStack 升级为 V2.2.2，之后，于 2009 年 12 月升级为 V2.3.0；2010 年 5 月，ZStack 升级为 V2.3.1。

1. ZStack 的结构

打开 ZStack 协议栈提供的示例工程，可以看到如图 1.17 所示的层次结构图。

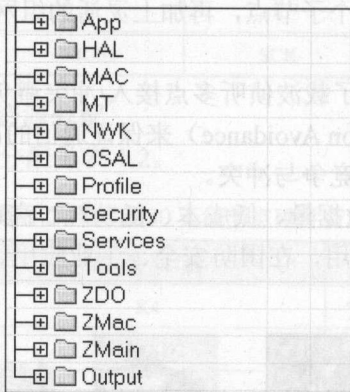


图 1.17 ZStack 软件结构图

应用中较多的是 HAL 层（硬件抽象层）和 App 层（用户应用），前者要针对具体的硬件进行修改，后者要添加具体的应用程序，工程每个目录的分析如下。

(1) App (Application Programming)：应用层目录，创建各种不同工程的区域，在这个目录中包含了应用层和项目的主要内容。

(2) HAL (Hardware(H/W)Abstraction Layer)：硬件层目录，包含有与硬件相关的配置和驱动及操作函数。

(3) MAC：MAC 层目录，包含了 MAC 层的参数配置文件及其 MAC 的 LIB 库的函数接口文件。

(4) MT (Monitor Test)：实现通过串口可控各层，与各层进行直接交互，同时可以将各层的数据通过串口连接到上位机，以便开发人员调试。

(5) NWK (ZigBee Network Layer)：网络层目录，含网络层配置参数文件及网络层库的函数接口文件。

(6) OSAL (Operating System Abstraction Layer)：协议栈的操作系统。

(7) Profile：AF (Application Framework) 层（应用构架）目录，包含 AF 层处理函数文件。Z-Stack 的 AF 层提供了开发人员建立一个设备描述所需的数据结构和辅助功能，是传入信息的终端多路复用器。

(8) Security：安全层目录，安全层处理函数，比如加密函数等。

(9) Services：地址处理函数目录，包括着地址模式的定义及地址处理函数。

(10) Tools：工程配置目录，包括空间划分及 ZStack 相关配置信息。

(11) ZDO (ZigBee Device Objects)：ZigBee 设备对象层 (ZDO) 提供了管理一个 ZigBee



设备的功能。ZDO 层的 API 为应用程序的终端提供了管理 ZigBee 协调器、路由器或终端设备的接口。这包括创建、查找和加入一个 ZigBee 网络，绑定应用程序终端，以及安全管理。

(12) ZMac: MAC 层目录，包括 MAC 层参数配置及 MAC 层 LIB 库函数回调处理函数。

(13) ZMain: 主函数目录，包括入口函数及硬件配置文件。

(14) Output: 输出文件目录，这个 EW8051 IDE 是自动生成的。

在 ZStack 协议栈中各层次具有一定的关系，如图 1.18 所示是 ZStack 协议栈的体系结构图。

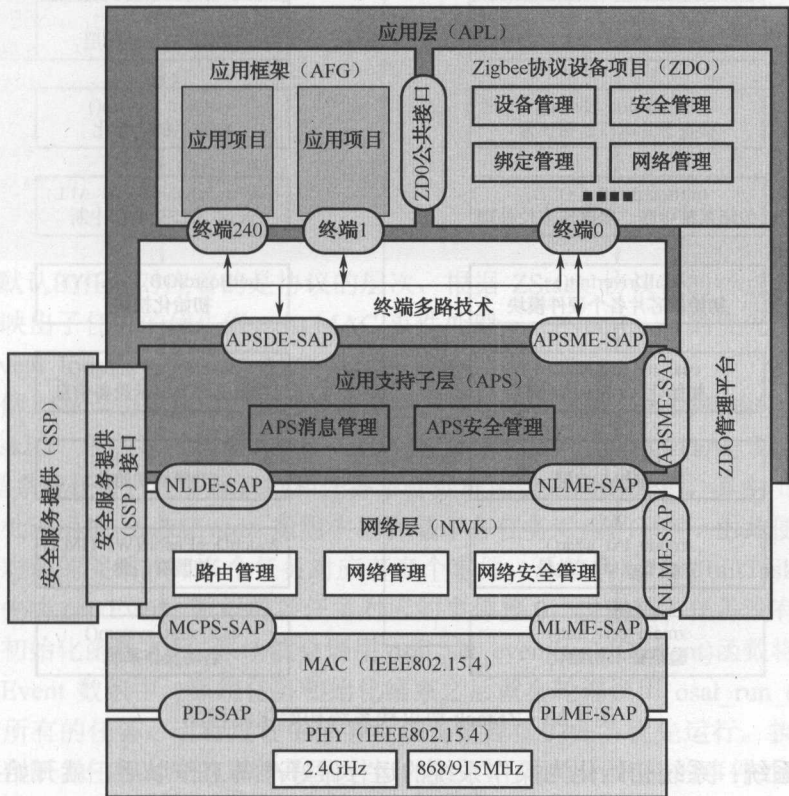


图 1.18 ZStack 协议栈的体系结构图

TI 公司的 Z-Stack 协议栈是一个基于轮转查询式的操作系统，它的 main 函数在 ZMain 目录下的 ZMain.c 中，该协议栈总体上来说，一共做了两件工作，一个是系统初始化，即由启动代码来初始化硬件系统和软件构架需要的各个模块，另外一个就是开始启动操作系统，如图 1.19 所示。

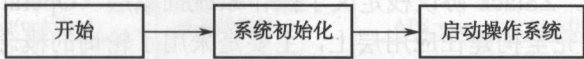


图 1.19 协议栈主要工作流程

2. 系统初始化

系统启动代码须要完成初始化硬件平台和软件架构所需的各个模块，微操作系统的运行做好准备工作，主要分为初始化系统时钟，检测芯片工作电压，初始化堆栈，初始化各个硬件模块，初始化 Flash 存储，形成芯片 MAC 地址，初始化非易失变量，初始化 MAC 层



协议，初始化应用帧层协议，初始化操作系统等 10 余部分，其具体流程图和对应的函数如图 1.20 所示。

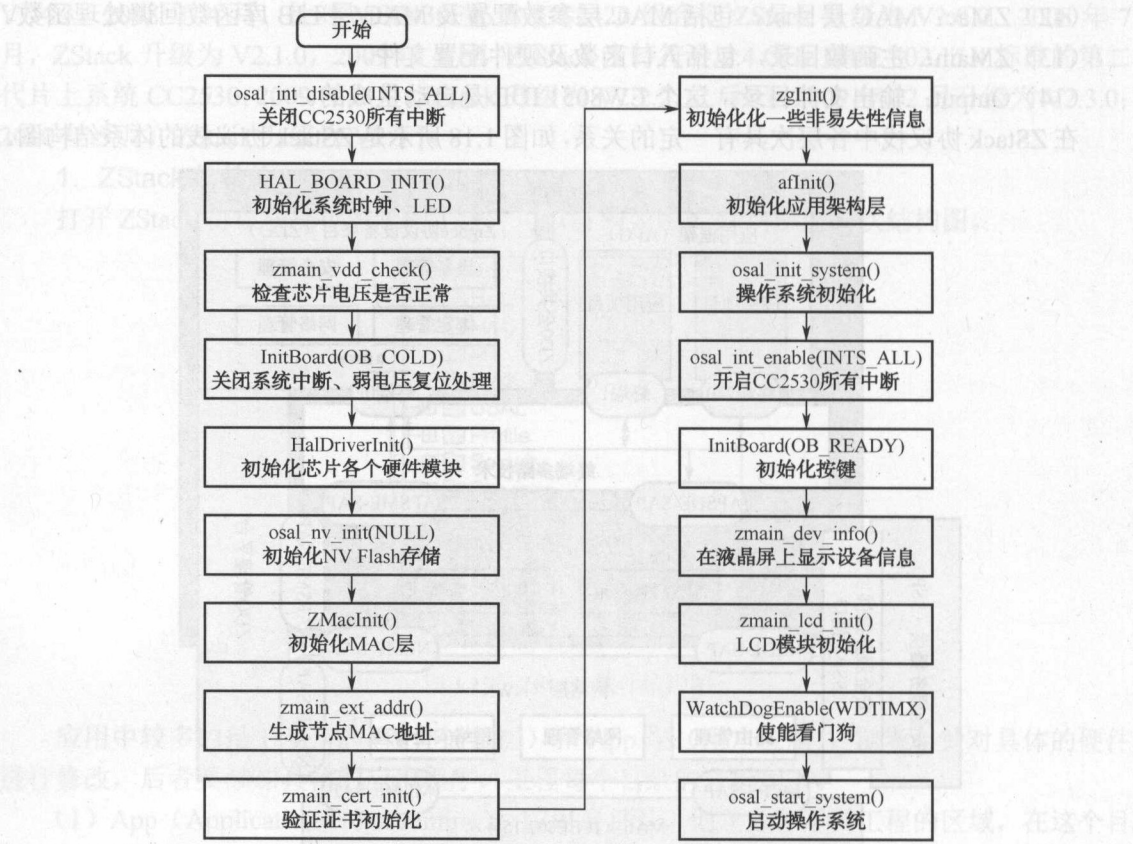


图 1.20 ZStack 协议栈系统初始化流程图

启动操作系统：系统初始化为操作系统的运行做好准备工作以后，就开始执行操作系统入口程序，并由此彻底将控制权交给操作系统，其实，启动操作系统实体只有一行代码。

```
osal_start_system();
```

该函数没有返回结果，通过将该函数一层层展开之后就知该函数其实就是一个死循环。这个函数就是轮转查询式操作系统的主体部分，他所做的就是不断地查询每个任务是否有事件发生，如果发生，执行相应的函数，如果没有发生，就查询下一个任务。

3. OSAL 调度管理

为了方便任务管理，ZStack 协议栈定义了操作系统抽象层（Operation System Abstraction Layer, OSAL）。OSAL 完全构建在应用层上，主要是采用了轮询的概念，并且引入了优先级。它的主要作用是隔离 ZStack 协议栈和特定硬件系统，开发者无须过多了解具体平台的底层，就可以利用操作系统抽象层提供的丰富工具实现各种功能，包括任务注册、初始化和启动、同步任务、多任务间的消息传递、中断处理、定时器控制、内存定位等。

OSAL 中判断事件发生是通过 tasksEvents[idx]任务事件数组来进行的。在 OSAL 初始化的时候，tasksEvents[]数组被初始化为零，一旦系统中有事件发生，就用 osal_set_event 函数把 tasksEvents[taskID]赋值为对应的事件。不同的任务有不同的 taskID，这样任务事件数组



tasksEvents 中就表示了系统中哪些任务存在没有处理的事件。然后就会调用各任务处理对应的事件，任务是 OSAL 中很重要的概念。任务通过函数指针来调用，参数有两个：任务标识符 (taskID) 和对应的事件 (event)。ZStack 中有 7 种默认的任务，它们存储在 taskArr 这个函数指针数组中。定义如下：

```
const pTaskEventHandlerFn tasksArr[] = {
    macEventLoop,
    nwk_event_loop,
    Hal_ProcessEvent,
    #if defined( MT_TASK )
        MT_ProcessEvent,
    #endif
    APS_event_loop,
    ZDApp_event_loop,
    SAPI_ProcessEvent
};
```

7 个事件默认的任务对应着的是协议的层次，根据 ZStack 协议栈的特点，这些任务从上到下的顺序反映出了任务的优先级，如 MAC 事件处理 macEventLoop 的优先级高于网路层事件处理 nwk_event_loop。

要深入理解 ZStack 协议栈中 OSAL 的调度管理关键是要理解任务的初始化 osalInitTasks()、任务标识符 taskID、任务事件数组 tasksEvents、任务事件处理函数 tasksArr 数组之间的关系。

图 1.21 是系统任务、任务标识符和任务事件处理函数之间的关系。其中 tasksArr 数组中存储了任务的处理函数，tasksEvents 数组中则存储了各任务对应的事件，由此便可得知任务与事件之间是多对多的关系，即多个任务对应着多个事件。系统调用 osalInitTasks()函数进行任务初始化时首先将 taskEvents 数组的各任务对应的事件置 0，也就是各任务没有事件。当调用了各层的任务初始化函数之后，系统就会调用 osal_set_event(taskID,event)函数将各层任务的事件存储到 taskEvent 数组中。系统任务初始化结束之后就会轮询调用 osal_run_system()函数开始运行系统中所有的任务，运行过程中任务标识符值越低的任务优先运行。执行任务的过程中，系统就会判断各任务对应的事件是否发生，若发生了则执行相应的事件处理函数。

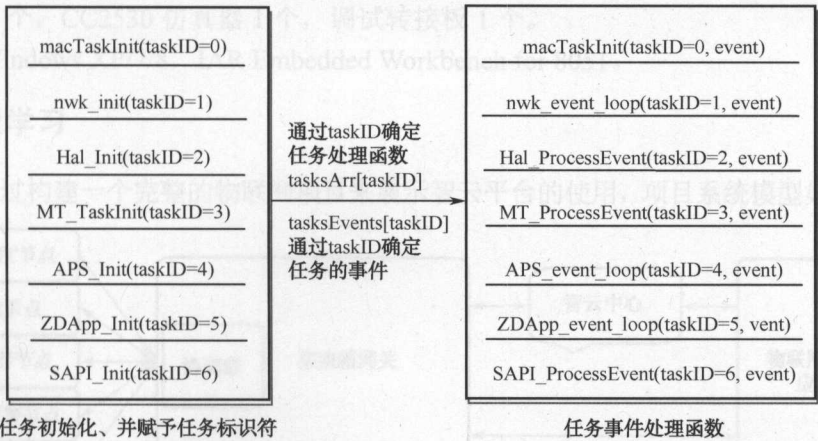


图 1.21 任务事件之间的关系

本章先介绍智云平台配置，并详细学习通信协议，重点学习硬件驱动开发、AndroidAPI开发和 WebAPI 开发，学会用调试工具进行开发，最后实现应用项目上传，开发者可以快速熟悉设计一个智云物联网应用项目。

2.1 任务 5：智云平台配置

2.1.1 学习目标

- 掌握智云平台硬件的部署；
- 了解智云网站项目及 ZCloudApp 的使用；
- 了解 ZCloudTools 工具的使用；
- 了解 ZCloudDemo 程序的使用。

2.1.2 开发环境

硬件：温度传感器 1 个，光敏传感器 1 个，继电器传感器 1 个，声光报警传感器 1 个，步进电机传感器 1 个，智云 Android 开发平台 1 个(默认为 S210 系列 Android 开发平台),CC2530 无线节点板 5 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051。

2.1.3 原理学习

本任务通过构建一个完整的物联网项目来展示智云平台的使用，项目系统模型如图 2.1 所示。

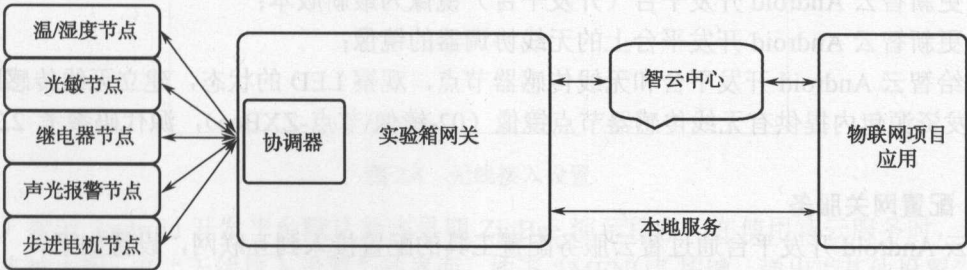


图 2.1 智云平台系统模型



(1) 协调器和温/湿度节点、光敏节点、继电器节点、声光报警节点、步进电机节点通过 ZigBee 无线传感网络联系在一起，其中协调器节点作为整个网络的汇集中心。

(2) 协调器与开发平台进行交互，通过开发平台上运行的服务程序将传感网与电信网或移动网进行连接，同时将数据推送给智云中心，也支持数据推送到本地局域网。

(3) 智云数据中心提供数据的存储服务、数据推送服务、自动控制服务等深度的项目接口，本地服务仅支持数据的推送服务。

(4) 物联网应用项目通过智云 API 进行具体应用的开发，能够实现对传感网内节点进行采集、控制、决策等。

2.1.4 开发内容

智云平台通过以下简单的几个步骤即可完成项目部署，如图 2.2 所示。

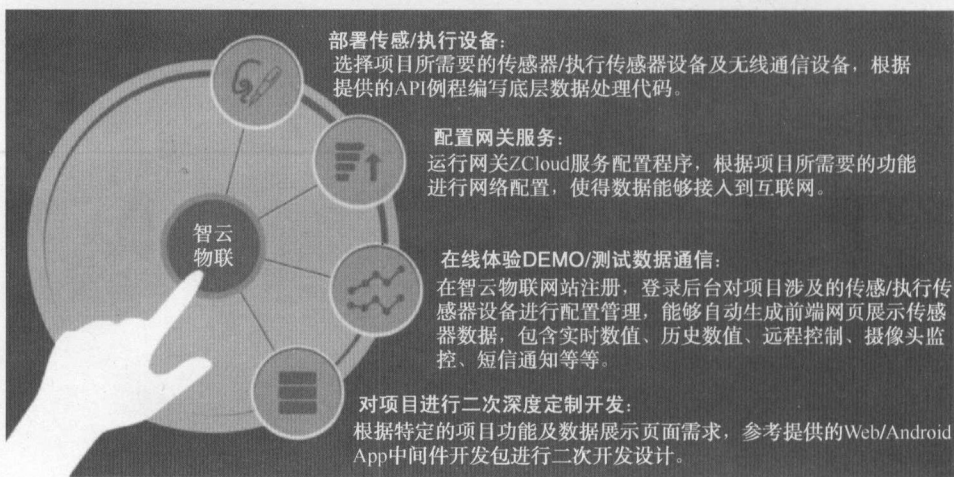


图 2.2 智云平台项目部署示意图

1. 部署传感/执行设备

智云平台硬件系统包括无线传感器节点和智云 Android 开发平台（开发平台），无线传感器节点通过 ZigBee 协议与智云 Android 开发平台的无线协调器构建无线传感网，然后通过智云 Android 开发平台内置的智云服务与移动网/电信网进行连接，通过上层应用进行采集与控制。其中，无线传感器节点硬件部署如下。

- 根据无线传感器节点所携带的传感器类型固化镜像；
- 更新智云 Android 开发平台（开发平台）镜像为最新版本；
- 更新智云 Android 开发平台上的无线协调器的镜像；
- 给智云 Android 开发平台和无线传感器节点，观察 LED 的状态，建立无线传感网络。

开发资源包内提供有无线传感器节点镜像（02-镜像\节点-ZXBee），源代码参考 2.3 节的分析。

2. 配置网关服务

智云 Android 开发平台通过智云服务配置工具的配置接入到互联网，设置如下。

(1) 将开发平台通过 3G、Wi-Fi、以太网等任意一种方式接入到互联网（若仅在局域网内使用，可不用连接到互联网），在 Android 开发平台的 Android 系统运行程序——智云服务配



置工具。

(2) 在用户账号、用户密钥栏输入正确的智云 ID/KEY，也可单击“扫一扫二维码”按钮，用摄像头扫描获得 ID/KEY 的二维码图片，自动填写 ID/KEY（若数据仅在局域网使用，可任意填写）。

(3) 服务地址为 zhiyun360.com，若使用本地搭建的智云数据中心服务，则填写正确的本地服务地址。

(4) 单击“开启远程服务”按钮，成功连接智云服务后则支持数据传输到智云数据中心；单击“开启本地服务”按钮，成功连接后智云服务将向本地进行数据推送，如图 2.3 所示。



图 2.3 配置网关服务

说明：智云服务配置工具配置之前需要对接入的节点进行设置。

(5) 在智云服务配置工具主界面，按下“MENU”按键，弹出“无线接入设置”菜单，单击进入菜单，在弹出的界面勾选“ZigBee 配置”选项（默认该服务会自动判别智云 Android 开发平台的串口设置，若需要更改则单击“ZigBee 配置”项，在弹出的菜单选择串口），设置成功后，提示服务已启动，如图 2.4 所示。

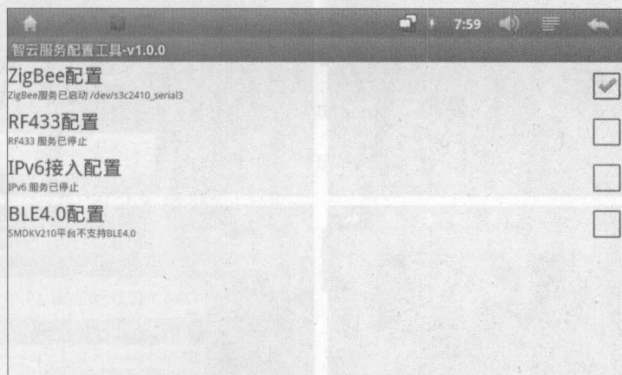


图 2.4 无线接入设置

(6) 智云 Android 开发平台默认兼容早期 ZigBee 演示程序，在使用智云服务时，须要确保串口未被占用，在“无线接入设置”的界面，按下“MENU”按键，弹出“其他设置”菜单，单击进入菜单，在弹出的界面将“启用 ZigBee 网关”选项关闭，如图 2.5 所示。

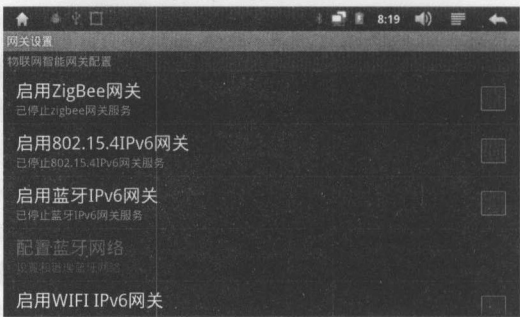


图 2.5 其他设置

3. 测试数据通信

智云物联开发平台提供了智云综合应用用于项目的演示及数据调试，安装 ZCloudTools 应用程序并对硬件设备进行演示及调试。

ZCloudTools 应用程序包含四大功能：网络拓扑及硬件控制、节点数据分析与调试、节点传感器历史数据查询、ZigBee 网络信息远程更新，主要操作演示界面如图 2.6 所示。



图 2.6 ZCloudTools 功能展示

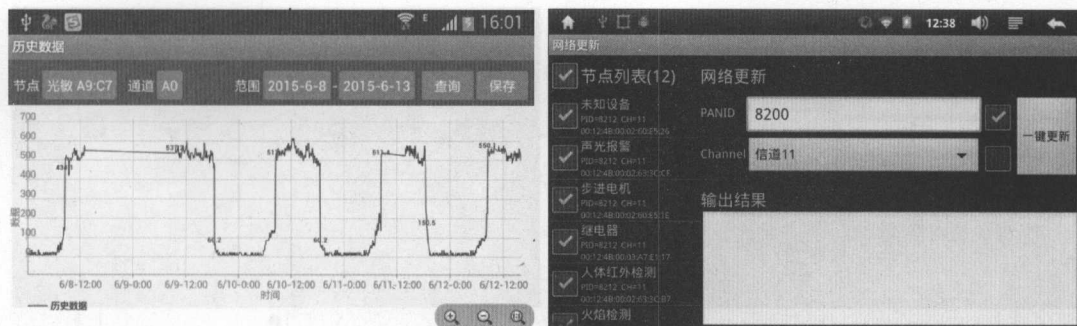


图 2.6 ZCloudTools 功能展示 (续)

说明：ZCloudTools 程序添加了默认智云 ID/KEY，可以接入在线的设备进行演示。

4. 在线体验 DEMO

智云物联开发平台提供了针对 Android 的应用组态 DEMO 程序，支持设备的动态添加、删除和管理。通过项目信息的导入，能够自动为设备生成特有属性功能：传感器进行历史数据曲线展示及实时数据的自动更新展示，执行器通过动作按钮进行远程控制且可对执行动作进行消息跟踪，摄像头可以通过动作按钮控制云台转动。无须编程即可完成不同应用项目的构建，如智能家居管理平台、智能农业管理平台……安装 ZCloudDemo 应用程序并对硬件设备进行演示及调试，相关参考截图如下。

(1) 导入配置文件。运行 ZCloudDemo 程序，按下菜单按键，在弹出的菜单项选择导入 ZCloudDemoV2.xml 文件，如图 2.7 所示。

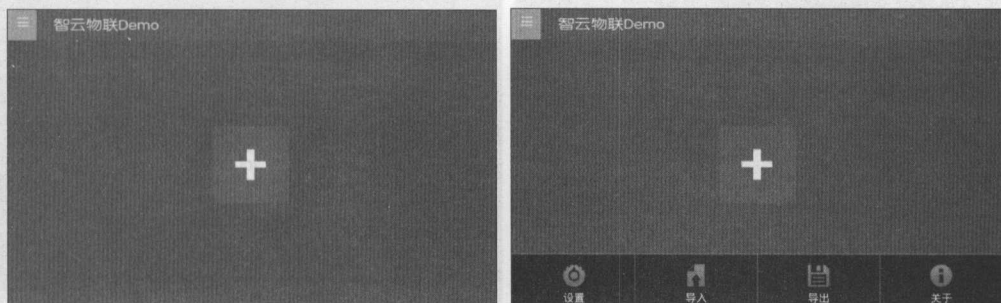


图 2.7 ZCloudDemo 配置文件导入

(2) 查看设备信息。导入成功后将自动生成所有设备列表模块，单击设备图标即可展示该设备的信息，部分截图如图 2.8 所示。

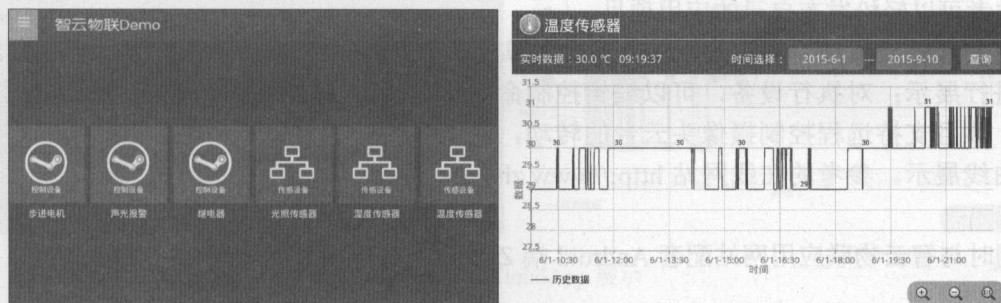


图 2.8 ZCloudDemo 设备信息查看

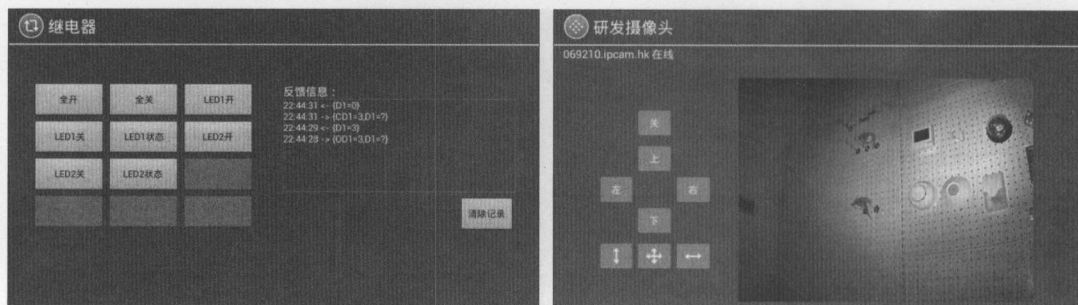


图 2.8 ZCloudDemo 设备信息查看（续）

（3）添加/删除设备。单击“+”图标可添加新的设备，长按设备图标弹出对话框提示是否编辑/删除设备，如图 2.9 所示。

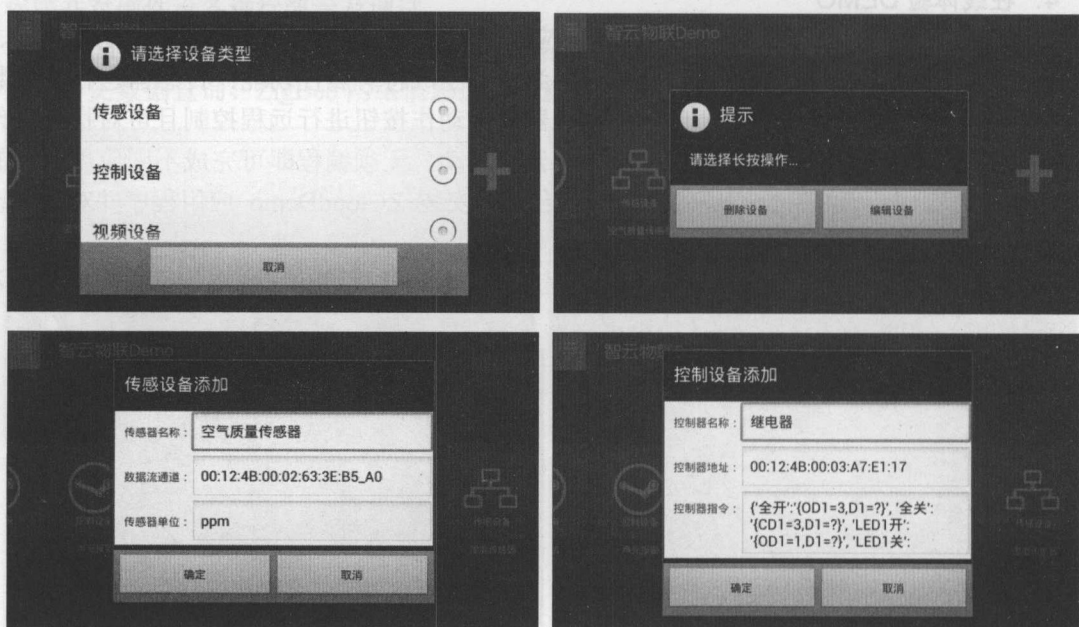


图 2.9 ZCloudDemo 设备添加/删除

5. 智云网站及 App

智云平台为开发者提供一个应用项目分享的应用网站 <http://www.zhiyun360.com>，通过注册开发者可以轻松发布自己的应用项目。

开发者的应用项目可以展示节点采集的实时在线数据、查询历史数据，并且以曲线的方式进行展示；对执行设备，可以编辑控制命令，远程控制设备；同时可以在线查阅视频图像，并且支持远程控制摄像头云台的转动，支持设置自动控制逻辑进行摄像头图片的抓拍并曲线展示。参考的在线网站 <http://www.zhiyun360.com/Home/Sensor?ID=15>，如图 2.10 所示。

同时与智云物联应用网站配套 Android 端 ZCloudApp 应用界面如图 2.11 所示。



图 2.10 智云网站展示

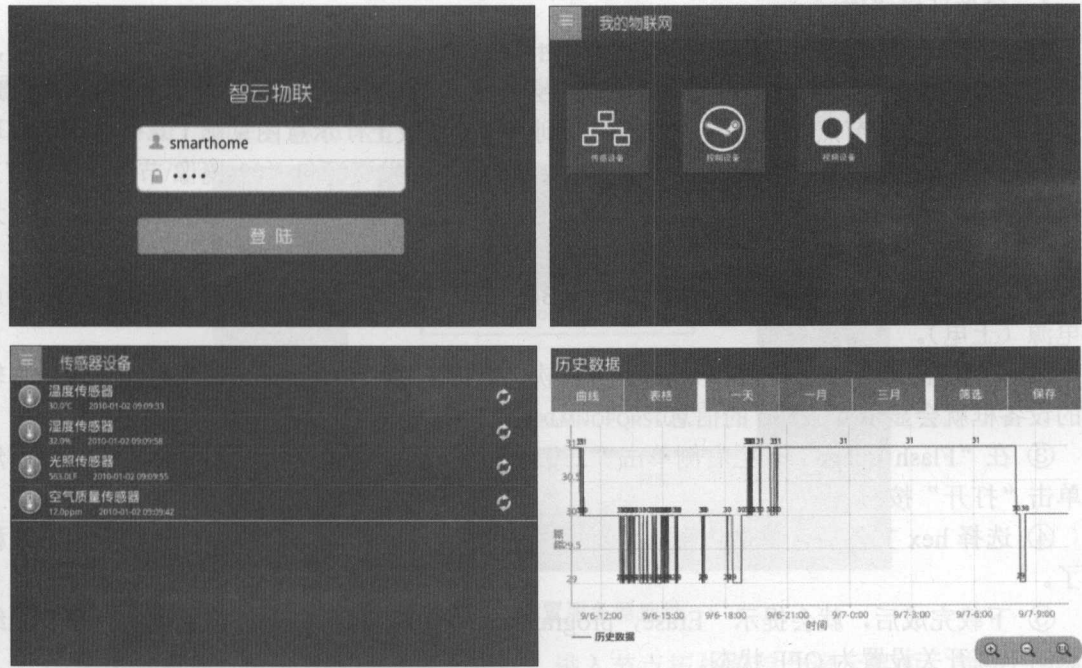


图 2.11 ZCloudApp 展示

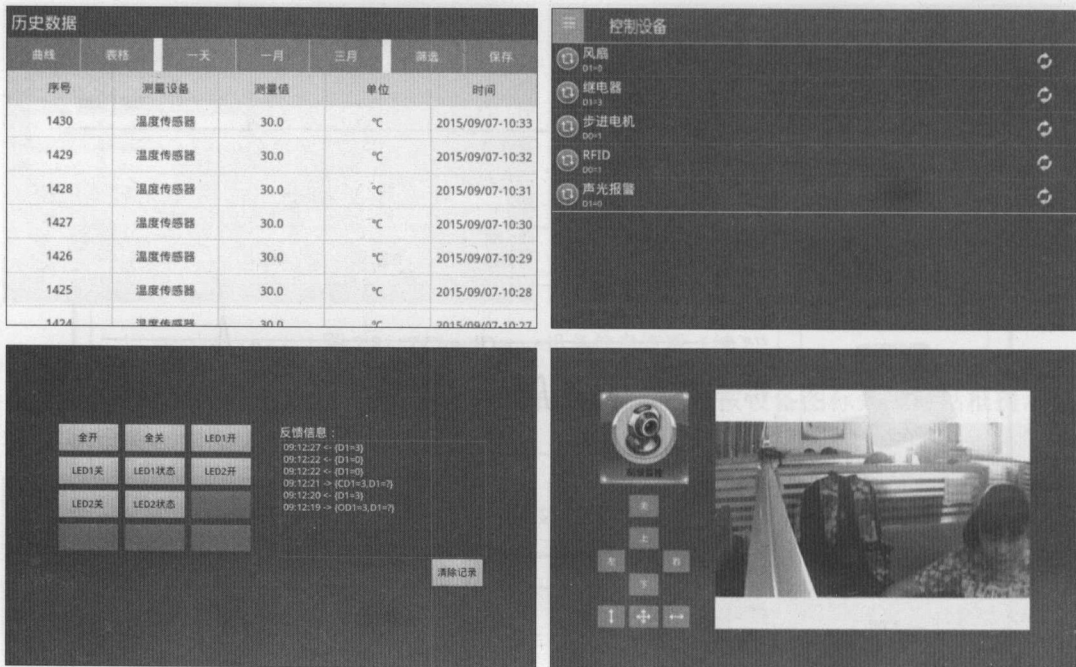


图 2.11 ZCloudApp 展示 (续)

2.1.5 开发步骤

1. 硬件环境搭建

(1) 准备一套 S210 系列 Android 开发平台，将无线协调器节点插入到对应的主板插槽，准备无线节点板（如果集成在开发平台主板上会集中供电，单独使用需要配合 5 V/3 A 电源适配器使用），将无线节点板和对应的传感器接到节点扩展板上，示意图见第 1 章硬件框图。

(2) 将 SmartRF Flash Programmer 下载 Hex 文件（开发资源包“02-镜像\节点-ZXBee”）固化到 ZigBee 节点中，如协调器、温湿度、继电器等节点（开发者根据已选购好的传感器为准），节点烧写步骤如下。

① 正确连接 CC2530 仿真器到 PC 和 ZXBee CC2530 节点板，打开 ZXBee CC2530 节点板电源（上电）。

② 运行 SmartRF Flash Programmer 仿真软件，按下 CC2530 仿真器的复位按键，仿真软件的设备框就会显示 CC2530 的信息。

③ 在“Flash image”一栏右侧单击“...”按钮选择温/湿度（协调器、继电器）.hex，然后单击“打开”按钮。

④ 选择 hex 文件后，单击仿真软件页面的“Perform actions”按钮，就可以开始下载程序了。

⑤ 下载完成后，就会提示“Erase, program and verify OK”信息，将无线协调器和无线节点的电源开关设置为 OFF 状态。

(3) 给 Android 上电，长按 Power 按键开机进入到 Android 系统。

(4) 根据开发需要，选用 Wi-Fi、以太网接口、3G 将开发平台连接互联网。

(5) 先给无线协调器上电，此时 D6 LED 灯开始闪烁，当正确建立好网络后，D6 LED 会



常亮。

(6) 当无线协调器建立好网络后，给 6 个无线节点上电，此时每个无线节点的 D6 LED 灯开始闪烁，直到加入到协调器建立的 ZigBee 网络中后，D6 LED 灯开始常亮。

(7) 当有数据包进行收发时，无线协调器和无线节点的 D7 LED 灯会闪烁。

2. 配置网关服务

按照 2.1.4 节方法配置网关服务。

3. ZCloudTools 功能演示

运行 ZCloudTools 用户控制程序，ZCloudTools 用户程序运行后就会进入如图 2.12 所示的页面。



图 2.12 ZCloudTools 程序入口界面

(1) 服务器地址和网关的设置。进入 ZcloudTools 主界面后，单击“MENU”键，选择“配置网关”菜单选项，输入服务地址 zhiyun360.com，输入用户账号和用户密钥（智云项目 ID/KEY），单击“确定”按钮保存，如图 2.13 所示。

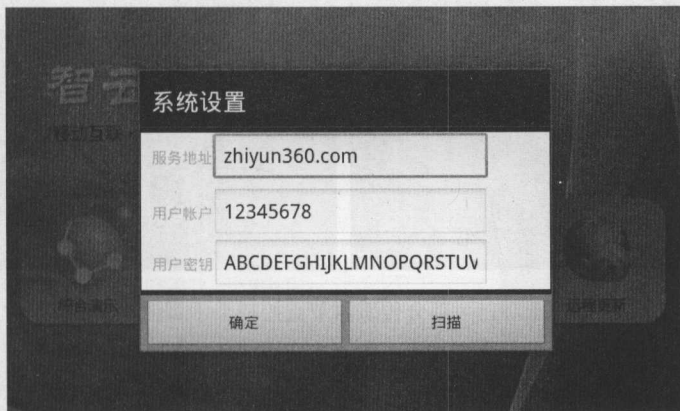


图 2.13 设置服务器地址

(2) 综合演示。单击“综合演示”图标，进入节点拓扑图综合演示界面，等待一段时间后，就会形成所有传感节点的拓扑图结构，包括协调器（红色）、路由节点（紫色）和终端节点（浅蓝色），如图 2.14 所示。

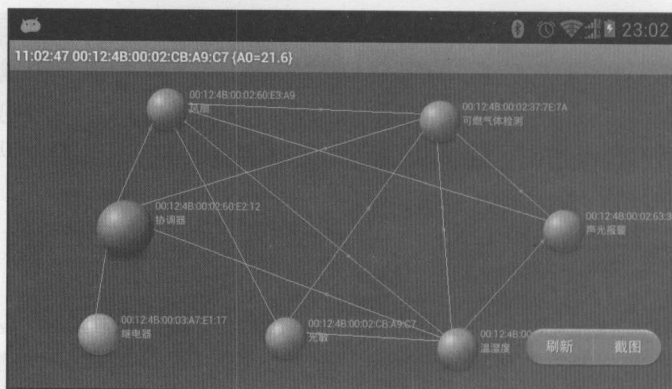


图 2.14 节点拓扑图结构

单击节点的图标就可以进入相应的节点控制页面，图 2.15 所示为部分传感器的操作页面。操作方法由开发者自行操作，采集类传感器以曲线形式显示采集到的值，安防类传感器检测到变化后会做出警报声并提示相关消息，控制类传感器可以直接控制相关的操作。



图 2.15 部分传感器节点控制显示页面

(3) 数据分析。单击“数据分析”图标，进入数据分析界面（在此以温湿度节点为例介绍调试过程）。单击节点列表中的“温湿度”选项，进入温湿度节点调试界面。输入调试指令“{A0=?,A1=?}”并发送，查询当前温湿度值，如图 2.16 所示。

输入调试指令“{V0=3}”并发送，修改主动上报时间间隔为 3 s，如图 2.17 所示。

输入调试指令“{OD1=1,D1=?}”，发送指令后，禁止温度值上报，调试信息窗口只显示当前湿度值，如图 2.18 所示。

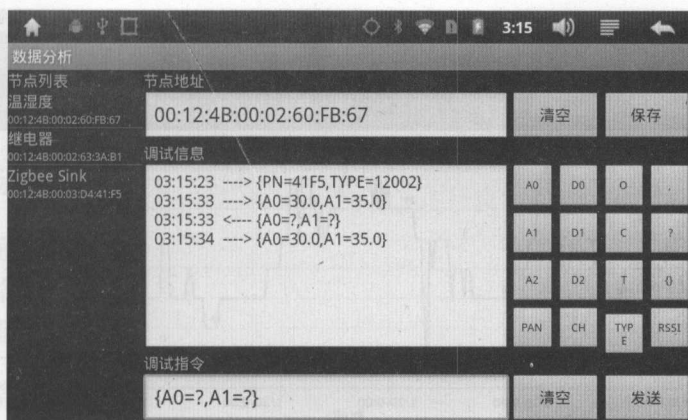


图 2.16 查询温湿度值

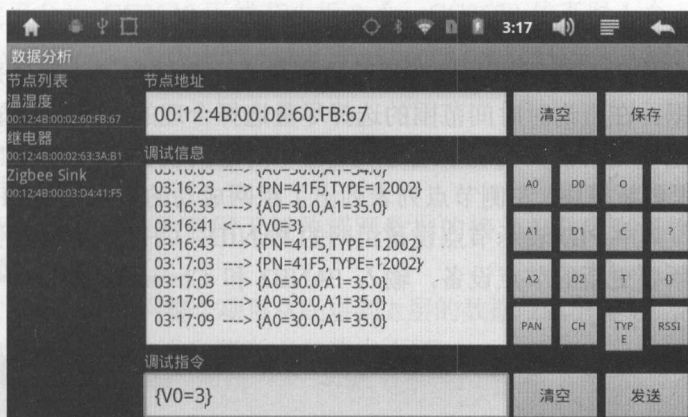


图 2.17 修改上报时间间隔

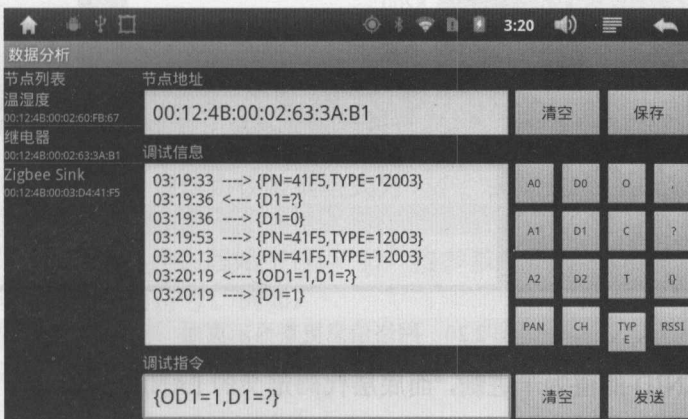


图 2.18 禁止温度值上报

(4) 历史数据。历史数据模块实现了获取指定设备节点某时间段的历史数据。单击“历史数据”图标进入历史数据查询功能模块，选择“温湿度”，通道选择 A0，时间范围选在“2015-1-1”至“2015-2-1”时间段，单击“查询”按钮，历史数据查询成功后会以曲线的形式显示在页面中，如图 2.19 所示。



图 2.19 历史数据查询显示页面

只有当开发平台连入互联网，并且在智云数据中心的存储有该传感器采集到的值时，才能够查询到历史数据。在查询时时间范围的选择尽量选择合理的时间进行查询。

(5) 远程更新。远程更新模块实现了通过发送命令对组网设备节点的 PANID 和 CHANNEL 进行更新，进入远程更新模块，左侧节点列表列出了组网成功的节点设备 (PID=8212 CH=11 <节点 MAC 地址>)，其中 PID 表示节点设备组网的 PANID，CH 表示其组网的 Channel。依次单击复选框，选择所要更新的节点设备，输入 PANID 和 Channel，单击“一键更新”按钮，执行更新，如图 2.20 所示。



图 2.20 网络信息更新显示页面

注意：此处 PANID 的值为十进制，而底层代码定义的 PANID 的值为十六进制，需要自行转换。示例如下：8200（十进制）= 0x2008（十六进制），通过“{PANID=8200}”命令将节点的 PANID 修改为 0x2008。

2.1.6 总结与拓展

搭建智云硬件环境，安装 ZCoudTools 软件应用进行演示，掌握智云平台的使用。



2.2 任务6：认识通信协议

2.2.1 学习目标

- 熟悉智云通信协议；
- 掌握协议格式定义；
- 掌握传感器的协议设计。

2.2.2 开发环境

硬件：温度传感器 1 个，继电器传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 2 个，CC2530 仿真器 1 个，调试转接板 1 个；

软件：Windows XP/7/8，IAR Embedded Workbench for 8051。

2.2.3 原理学习

1. 智云通信协议说明

智云物联云服务平台支持物联网无线传感网数据的接入，并定义了物联网数据通信的规范——智云数据通信协议。

智云数据通信协议对物联网整个项目从底层到上层的数据段做出了定义，该协议有以下特点。

- 数据格式的语法简单，语义清晰，参数少而精；
- 参数命名合乎逻辑，见名知义，变量和命令的分工明确；
- 参数读写权限分配合理，可以有效抵抗不合理的操作，能够在最大程度上确保数据安全；
- 变量能对值进行查询，可以方便应用程序调试；
- 命令是对位进行操作，能够避免内存资源浪费。

总之，智云数据通信协议在物联网无线传感网中易于应用和推广，开发者也容易在其基础上根据需求进行定制、扩展和创新。

2. 智云通信协议

(1) 通信协议数据格式。通信协议数据格式为

```
{[参数]=[值],[参数]=[值],……}
```

其中，每条数据以“{”作为起始字符；“}”内参数多个条目以“,”分隔；通信协议数据格式中的字符均为英文半角符号。例如：

```
{CD0=1,D0=?}
```

(2) 通信协议参数说明。通信协议参数说明如下。

● 参数名称定义为：

✧ 变量：A0~A7、D0、D1、V0~V3。

✧ 命令：CD0、OD0、CD1、OD1。

✧ 特殊参数：ECHO、TYPE、PN、PANID、CHANNEL。

● 变量可以对值进行查询，如“{A0=?}”。

● 变量 A0~A7 在物联网云数据中心可以存储保存为历史数据。



- 命令是对位进行操作。

具体参数解释如下：

① A0~A7：用于传递传感器数值或者携带的信息量，权限为只能通过赋值“？”来进行查询当前变量的数值，支持上传到物联网云数据中心存储，示例如下。

- 温湿度传感器采用 A0 表示温度值，A1 表示湿度值，数值类型为浮点型 0.1 精度；
- 火焰报警传感器采用 A0 表示警报状态，数值类型为整型，固定为 0（未检测到火焰）或者 1（检测到火焰）；
- 高频 RFID 模块采用 A0 表示卡片 ID 号，数值类型为字符串。

智云通信协议数据格式为：

```
{参数=值,参数=值,.....}
```

即用一对大括号“{}”包含每条数据，“{}”内参数如果有多个条目，则用“,”进行分隔，例如：

```
{CD0=1,D0=?}。
```

② D0：D0 的 Bit0~Bit7 分别对应 A0~A7 的状态（是否主动上传状态），权限为只能通过赋值“？”来进行查询当前变量的数值，0 表示禁止上传，1 表示允许主动上传，示例如下。

- 温湿度传感器：A0 表示温度值，A1 表示湿度值，D0=0 表示不上传温度和湿度信息，D0=1 表示主动上传温度值，D0=2 表示主动上传湿度值，D0=3 表示主动上传温度和湿度值。
- 火焰报警传感器：A0 表示警报状态，D0=0 表示不检测火焰，D0=1 表示实时检测火焰。
- 高频 RFID 模块：A0 表示卡片 ID 号，D0=0 表示不上报卡号，D0=1 表示运行刷卡响应上报 ID 卡号。

③ CD0/OD0：对 D0 的位进行操作，CD0 表示位清零操作，OD0 表示位置一操作，示例如下。

- 温湿度传感器：A0 表示温度值，A1 表示湿度值，CD0=1 表示关闭 A0 温度值的主动上报。
- 火焰报警传感器：A0 表示警报状态，OD0=1 表示开启火焰报警监测，当有火焰报警时，会主动上报 A0 的数值。

④ D1：D1 表示控制编码，权限为只能通过赋值“？”来进行查询当前变量的数值，开发者根据传感器属性来自定义功能，示例如下。

- 温湿度传感器：D1 的 Bit0 表示电源开关状态，例如，D1=0 表示电源处于关闭状态，D1=1 表示电源处于打开状态。
- 继电器：D1 的 Bit 表示各路继电器状态，例如，D1=0 关闭两路继电器 S1 和 S2，D1=1 开启继电器 S1，D1=2 开启继电器 S2，D1=3 开启两路继电器 S1 和 S2。
- 风扇：D1 的 Bit0 表示电源开关状态，Bit1 表示正转反转，例如，D1=0 或者 D1=2 风扇停止转动（电源断开），D1=1 风扇处于正转状态，D1=3 风扇处于反转状态。
- 红外电器遥控：D1 的 Bit0 表示电源开关状态，Bit1 表示工作模式/学习模式，例如，D1=0 或者 D1=2 表示电源处于关闭状态，D1=1 表示电源处于开启状态且为工作模式，D1=3 表示电源处于开启状态且为学习模式。

⑤ CD1/OD1：对 D1 的位进行操作，CD1 表示位清零操作，OD1 表示位置一操作。

⑥ V0~V3：用于表示传感器的参数，开发者根据传感器属性自定义功能，权限为可读写，



示例如下。

- 温湿度传感器：V0 表示自动上传数据的时间间隔。
- 风扇：V0 表示风扇转速。
- 红外电器遥控：V0 表示红外学习的键值。
- 语音合成：V0 表示需要合成的语音字符。

⑦ 特殊参数：ECHO、TYPE、PN、PANID、CHANNEL。

ECHO：用于检测节点在线的指令，将发送的值进行回显，比如发送“{ECHO=test}”，若节点在线则回复数据“{ECHO=test}”。

TYPE：表示节点类型，该信息包含了节点类别、节点类型、节点名称，权限为只能通过赋值“?”来进行查询当前值。TYPE 的值由 5 个 ASCII 字节表示。例如，11001，第 1 字节表示节点类别（1 表示 ZigBee，2 表示 RF433，3 表示 Wi-Fi，4 表示 BLE，5 表示 IPv6，9 表示其他）；第 2 字节表示节点类型（0 表示汇集节点，1 表示路由/中继节点，2 表示终端节点）；第 3、4、5 字节合起来表示节点名称（编码开发者自定义）。ZXBeeEdu 系列教学节点 Type 类型定义如表 2.1 所示。

表 2.1 传感器的参数标识列表

节点编码	节点名称	节点编码	节点名称
000	协调器	020	直流电机传感器
001	光敏传感器	021	紧急按钮传感器
002	温湿度传感器	022	数码管传感器
003	继电器传感器	023	低频 RFID 传感器
004	人体红外检测	024	防水温度传感器
005	可燃气体检测	025	红外避障传感器
006	步进电机传感器	026	干簧门磁传感器
007	风扇传感器	027	红外对射传感器
008	声光报警传感器	028	二氧化碳传感器
009	空气质量传感器	029	颜色识别传感器
010	振动传感器	030	九轴自由度传感器
011	高频 RFID 传感器	031	一氧化碳传感器
012	三轴加速度传感器	100	红外遥控传感器
013	噪声传感器	101	流量计数传感器
014	超声波测距传感器	102	粉尘传感器
015	酒精传感器	103	土壤湿度传感器
016	触摸感应传感器	104	火焰识别传感器
017	雨滴/凝露传感器	105	语音识别传感器
018	霍尔传感器	106	语音合成传感器
019	压力传感器	107	指纹识别传感器

PN（仅针对 ZigBee/802.15.4 IPv6 节点）：表示节点的上行节点地址信息和所有邻居节点地址信息，权限为只能通过赋值“?”来进行查询当前值。



PN 的值为上行节点地址和所有邻居节点地址的组合。其中每 4 个字节表示一个节点地址后 4 位，第一个 4 字节表示该节点上行节点后 4 位，第 2~n 个 4 字节表示其所有邻居节点地址后 4 位。

PANID：表示节点组网的标志 ID，权限为可读写，此处 PANID 的值为十进制，而底层代码定义的 PANID 的值为十六进制，需要自行转换。例如，8200（十进制）= 0x2008（十六进制），通过“{PANID=8200}”命令将节点的 PANID 修改为 0x2008。PANID 的取值范围为 1~16383。

CHANNEL：表示节点组网的通信通道，权限为可读写，此处 CHANNEL 的取值范围为十进制数 11~26。例如，通过命令“{CHANNEL=11}”将节点的 CHANNEL 修改为 11。

在实际应用中可能硬件接口会比较复杂，例如一个无线节点携带多种不同类型传感器数据，下面以一个示例来进行解析。

例如，某个设备具备以下特性，一个燃气检测传感器、一个声光报警装置、一个排风扇，要求由如下功能：

- 设备可以开关电源；
- 可以实时上报燃气浓度值；
- 当燃气达到一定峰值，声光报警器会报警，同时排风扇会工作；
- 据燃气浓度的不同，报警声波频率和排风扇转速会不同。

根据该需求，定义协议如表 2.2 所示。

表 2.2 复杂数据通信设备协议定义

传 感 器	属 性	参 数	权限	说 明
复杂设备	燃气浓度值	A0	R	燃气浓度值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示燃气浓度上传状态，OD0/CD0 进行状态控制
	开关状态	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示设备电源状态，Bit1 表示声光报警状态，Bit2 表示排风扇状态，OD1/CD1 进行状态控制
	上报间隔	V0	RW	修改主动上报的时间间隔
	声光报警声波频率	V1	RW	修改声光报警声波频率
	排风扇转速	V2	RW	修改排风扇转速

复杂的应用都是在简单的基础上进行的组合和叠加，不同传感器的组合可以实现复杂的应用功能。

3. 节点协议定义

传感器的智云通信协议参数定义可以如如表 2.3 所示。

表 2.3 传感器参数定义及说明

传 感 器	属 性	参 数	权限	说 明
温湿度	温度值	A0	R	温度值，浮点型：0.1 精度
	湿度值	A1	R	湿度值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示温度上传状态、Bit1 表示湿度上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔



续表

传 感 器	属 性	参 数	权限	说 明
光敏/空气质量/ 超声波/大气压力/ 酒精/雨滴/防水温 度/流量计数	数值	A0	R	数值, 浮点型: 0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔
三轴	X 值	A0	R	X 值, 浮点型: 0.1 精度
	Y 值	A1	R	Y 值, 浮点型: 0.1 精度
	Z 值	A2	R	Z 值, 浮点型: 0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示 X 值上传状态、Bit1 表示 Y 值上传状态、Bit2 表示 Z 值上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔
可燃气体/火焰/ 霍尔/人体红外/噪 声/振动/触摸/紧急 按钮/红外避障/土 壤湿度	数值	A0	R	数值, 0 或者 1 变化
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态
继电器	继电器开合	D1(OD1/CD1)	R(W)	D1 的 Bit 表示各路继电器开合状态, OD1 为开、CD1 为合
风扇	电源开关	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示电源状态, Bit1 表示正转/反转
	转速	V0	RW	表示转速
声光报警	电源开关	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示电源状态, OD1 为上电、CD1 为关电
	频率	V0	RW	表示发声频率
步进电机	转动状态	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示转动状态, Bit1 表示正转/反转 X0: 不转, 01: 正转, 11: 反转
	角度	V0	RW	表示转动角度, 0 表示一直转动
直流电机	转动状态	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示转动状态, Bit1 表示正转/反转 X0: 不转, 01: 正转, 11: 反转
	转速	V0	RW	表示转速
红外电器遥控	状态开关	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示工作模式/学习模式, OD1=1 为学习模式、CD1=1 为工作模式
	键值	V0	RW	表示红外键值
高频 RFID/低频 RFID	ID 卡号	A0	R	ID 卡号, 字符串
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示允许识别
语音识别	语音指令	A0	R	语音指令, 字符串, 不能主动去读取
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示允许识别并发送读取的语音指令
数码管	显示开关	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示是否显示码值
	码值	V0	RW	表示数码管码值
语音合成	合成开关	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示是否合成语音
	合成字符	V0	RW	表示需要合成的语音字符



续表

传 感 器	属 性	参 数	权限	说 明
指纹识别	指纹指令	A0	R	指纹指令，数值表示指纹编号，0 表示识别失败
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示允许识别

2.2.4 开发内容

ZCloudTools 软件提供了通信协议测试工具，进入程序的“数据分析”功能模块就可以测试 ZXBee 协议了。

数据分析模块实现了获取指定设备节点上传的数据信息，并通过发送指令实现对节点状态的获取以及控制执行。进入数据分析模块，左侧列表会依次列出网关下的组网成功的节点设备，如图 2.21 所示。

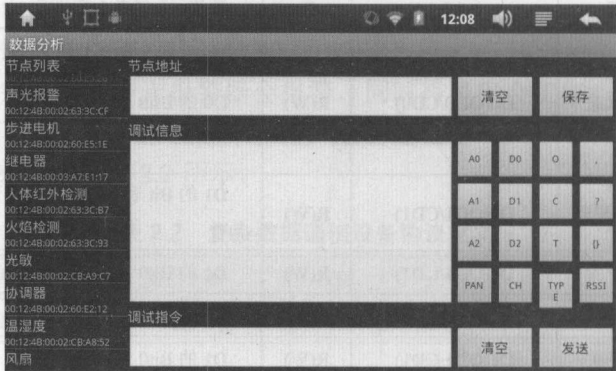


图 2.21 ZXBee 协议测试工具

单击节点列表中的某个节点，如继电器，ZCloudTools 自动将该节点的 MAC 地址填充到节点地址文本框中，并获取该节点所上传的数据信息显示在调试信息文本框中，如图 2.22 所示。

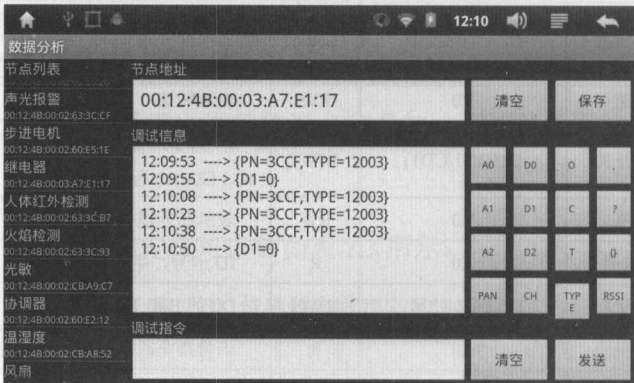


图 2.22 测试举例

也可通过输入命令查询继电器的状态、控制继电器转动等。例如，“{D1=?}”查询继电器状态，“{OD1=1,D1=?}”打开继电器，“{CD1=1,D1=?}”关闭继电器，如图 2.23 所示。

本任务以温湿度传感器和继电器传感器为例学习智云通信协议，节点温湿度传感器和继



电器传感器协议定义如表 2.4 所示。



图 2.23 测试举例

表 2.4 传感器参数定义及说明

传 感 器	属 性	参 数	权 限	说 明
温湿度	温度值	A0	R	温度值，浮点型：0.1 精度
	湿度值	A1	R	湿度值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示温度上传状态、Bit1 表示湿度上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔
继电器	继电器开合	D1(OD1/CD1)	R(W)	D1 的 Bit 表示各路继电器开合状态，OD1 为开、CD1 为合

在后续 2.5 节将介绍基于 Web 的调试工具的使用，将提供更丰富的调试功能。

2.2.5 开发步骤

以温湿度节点和继电器节点为例介绍通信协议。

- 按照任务 4 中的开发步骤方法，将温湿度节点、继电器节点、协调器节点（智云 Android 开发平台板载）的镜像文件固化到节点中；
- 准备一台智云 Android 开发平台；
- 参考 2.1 节内容，构建形成无线传感网络；
- 参考 2.1 节内容和步骤，对智云 Android 开发平台进行配置，确保智云网络连接成功；
- 运行 ZCloudTools 工具对节点进行调试。

单击“数据分析”图标，进入数据分析界面，单击节点列表中的“温湿度”，进入温湿度节点调试界面。输入调试指令“{A0=?,A1=?}”并发送，查询当前温湿度值，如图 2.24 所示。

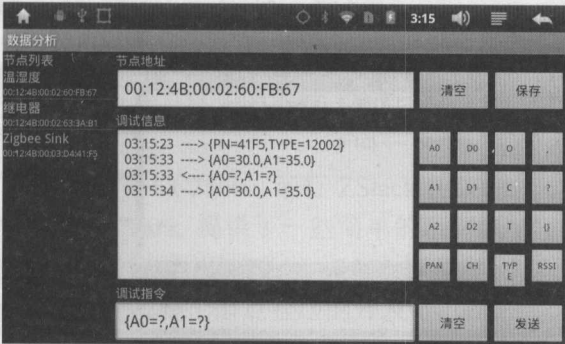


图 2.24 查询温湿度值



输入调试指令“{V0=3}”并发送，修改主动上报时间间隔为3s，如图2.25所示。

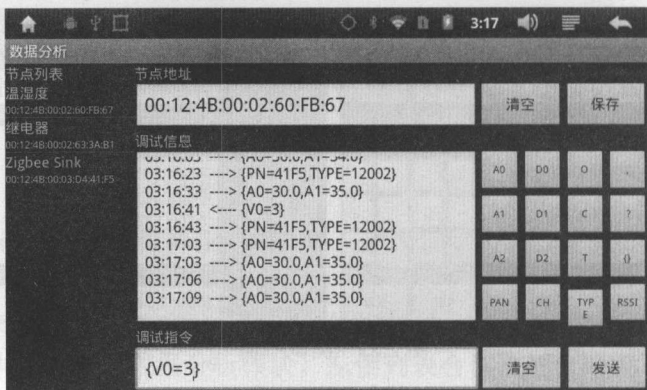


图 2.25 修改上报时间间隔

输入调试指令“{CD0=1}”，发送指令后，禁止温度值上报，调试信息窗口只显示当前湿度值，如图2.26所示。

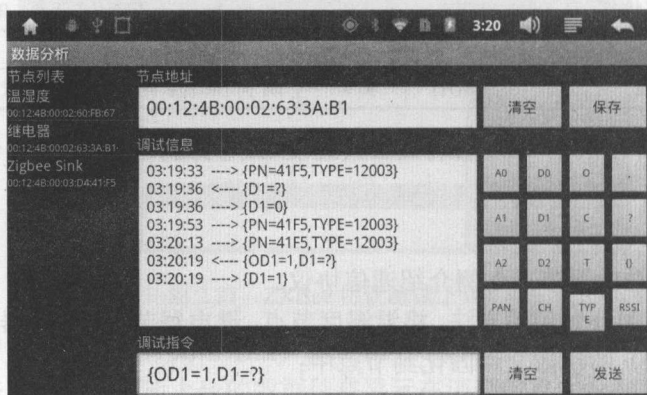


图 2.26 禁止温度值上报

单击节点列表中的“继电器”，进入继电器节点调试界面。输入调试指令“{D1=?}”并发送，查询当前继电器状态值，如图2.27所示。

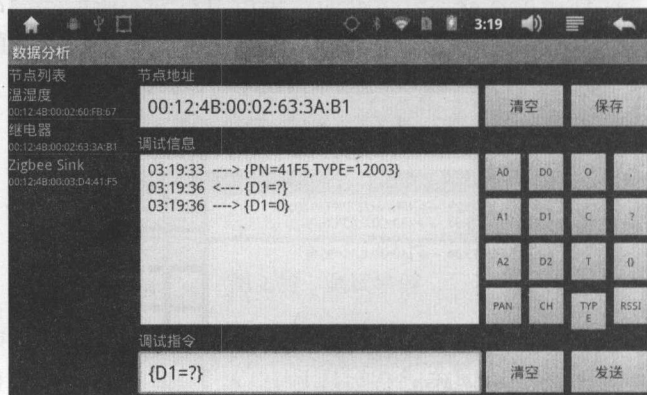


图 2.27 查询继电器状态值



输入调试指令“{CD0=1}”并发送，修改继电器状态值为1（即“开”状态）并查询当前继电器状态值，指令成功执行后会听到继电器开合的声音，执行结果如图 2.28 所示。

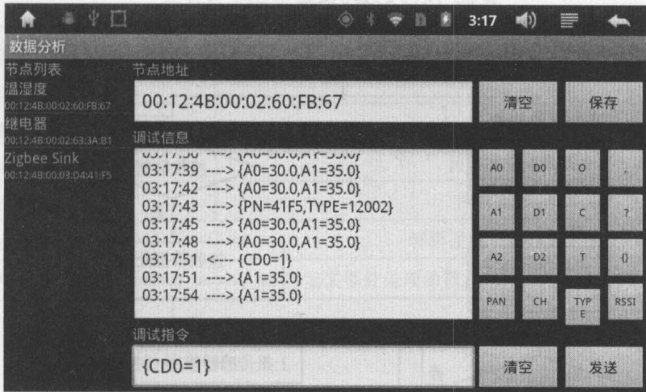


图 2.28 修改继电器状态值

2.2.6 总结与拓展

选择若干传感器/执行器，构建一套智能家居系统，并设计协议表格。

2.3 任务 7：硬件驱动开发

2.3.1 学习目标

- 熟悉 ZigBee 和智云通信协议；
- 掌握基于 ZigBee 和 CC2530 节点的硬件驱动开发。

2.3.2 开发环境

硬件：光敏传感器 1 个，人体红外传感器 1 个，继电器传感器 1 个，智云 Android 开发平台 1 个（S210 系列 Android 开发平台），CC2530 无线节点板 3 个，CC2530 仿真器 1 个，调试转接板 1 个；

软件：Windows XP/7/8，IAR Embedded Workbench for 8051。

2.3.3 原理学习

关于该节的详细开发过程请参考另外一本书籍《物联网平台开发及应用——基于 CC2530 和 ZigBee》，在这里直接介绍其应用。

无线节点处理器采用 TI 公司的 CC2530，运行 ZStack 协议栈，它为 CC2530 节点提供基于操作系统的无线自组网功能。ZStack 提供了一些简单的示例程序供开发者进行学习，其中 SimpleApp 工程是基于 SAPI 框架进行应用的开发，SAPI 接口实现了对应用的简单封装，开发者只需要要实现部分接口函数即可完成整个节点程序的开发，无线节点示例程序均基于 SAPI 框架开发，详细的程序流程图如图 2.29 所示。

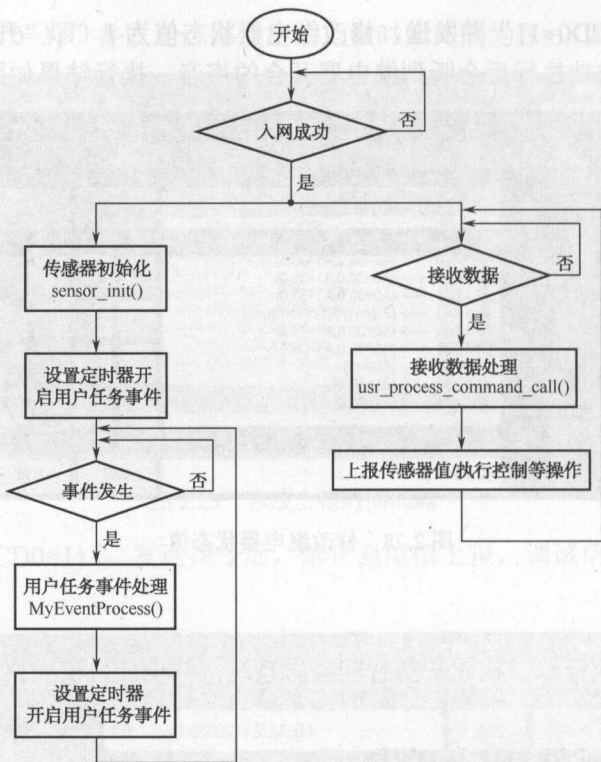


图 2.29 无线节点流程图

其中 SAPI 应用接口在 AppCommon.c 文件中实现，其中主要的几个函数如下：

void zb_HandleOsaiEvent(uint16 event)
(event & ZB_ENTRY_EVENT) 事件: 1.判断/设置节点类型（可按键修改节点类型） 2.设置网络信号LED（D6）灯闪烁 3.调用sensor_init()函数进行传感器初始化
(event & 0x000F) 事件: 1.进入用户自定义事件（0x0001~0x000F），该函数在UserApp.c文件内MyEventProcess()函数实现
void zb_StartConfirm(uint8 status)
检测是否正确入网，入网成功后LED（D6）灯长亮
void zb_ReceiveDataIndication(uint16 source, uint16 command, uint16 len, uint8 *pData)
处理接收到的无线数据包
static void process_package(char*pkg, int len)
处理并分解ZXBee无线数据包，提取命令键值对
static int _process_command_call(char*ptag, char*pval, char*pout)
1.处理ZXBee无线数据包内的命令键值对（通用命令） 2.调用usr_process_command_call(ptag, pval, pout)函数处理用户命令



智云平台为 ZigBee ZStack 协议栈上层应用提供分层的软件设计结构，将传感器的私有操作部分封装到 UserApp.c 文件中，详细函数说明如表 2.5 所示。

表 2.5 传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	传感器硬件初始化
sensor_update()	传感器数据定时上报
sensor_check()	传感器报警状态实时监测
sensor_control()	传感器/执行器控制函数
usr_process_command_call()	解析接收到的传感器控制命令函数
MyEventProcess()	自定义事件处理函数，启动定时器触发事件 MY_REPORT_EVT

2.3.4 开发内容

节点按功能可划分为采集类节点、报警类节点和控制类节点。

采集类传感器主要包括光敏传感器、温湿度传感器、可燃气体传感器、空气质量传感器、酒精传感器、超声波测距传感器、三轴加速度传感器、压力传感器、雨滴传感器等，主要用于采集环境值。

报警类传感器主要包括触摸开关传感器、人体红外传感器、火焰传感器、霍尔传感器、红外避障传感器、RFID 传感器、语音识别传感器等，主要用于检测外部环境并报警。

控制类传感器主要包括继电器传感器、步进电机传感器、风扇传感器、红外遥控传感器等，主要用于控制设备的状态。

1. 采集类传感器

光敏传感器主要采集光照值，ZXBee 协议定义如表 2.6 所示，光敏传感器程序逻辑驱动开发设计如图 2.30 所示。

表 2.6 光敏传感器智云通信协议定义

传感器	属性	参数	权限	说明
光敏传感器	数值	A0	R	数值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔

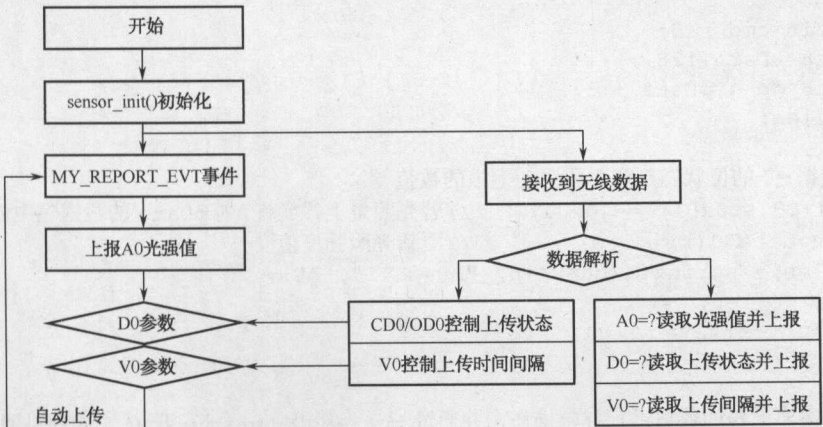


图 2.30 光敏传感器程序逻辑



程序实现过程如下。

(1) 在 `UserApp.h` 文件中编写以下代码。

```
#define MY_REPORT_EVT 0x0001           //定义事件 MY_REPORT_EVT
#define NODE_NAME "001"                //定义传感器参数标识
#define NODE_CATEGORY 1                //定义传感器类型
```

(2) 在 `UseApp.c` 文件中实现光敏传感器的初始化 `sensor_init()`：初始化传感器最基本的是配置选择寄存器和方向寄存器，此外还要启动一个定时器来触发事件 `MY_REPORT_EVT`。具体的代码实现如下。

```
//初始化传感器
void sensor_init(void)
{
    //配置 P0_1 端口为输入，且配置为外设功能 ADC:A1
    SENSOR_SEL |= SENSOR_BIT;
    SENSOR_DIR &= ~(SENSOR_BIT);

    //启动定时器，触发事件：MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10) * 1000));
}
```

(3) 在 `UseApp.c` 文件中实现自定义事件处理函数 `MyEventProcess(event)`，代码实现如下。

```
//自定义事件处理
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        //主动上报数据
        sensor_update();
        //启动定时器，触发事件：MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval * 1000));
    }
}
```

上述代码中调用了函数 `sensor_update()` 来上报采集到的数据，具体的代码实现如下。

```
//处理主动上报的数据
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01) { //若光照量上报允许，则 pData 的数据包中添加光照量数据
        updateA0();           //更新光照强度值
        len = sprintf((char*)p, "A0=%.1f", A0);
        p += len;
        *p++ = ',';
    }

    //将需要上传的数据进行打包操作，并通过 zb_SendDataRequest() 发送到协调器
    if (p - pData > 1) {
```




```

pData[0] = '{';
p[0] = 0;
p[-1] = '}';

zb_SendDataRequest(0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                    AF_DEFAULT_RADIUS );
HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK );           //通信 LED 闪烁一次
}
}

```

上述代码中调用了函数 `updateA0()` 来更新光照值，函数 `updateA0()` 的代码实现如下。

//更新光敏传感器采集的光照值

```
float updateA0(void)
```

```

{
    uint16 adcValue;

    //读取 ADC:A1 采集的电压量
    adcValue = HalAdcRead(HAL_ADC_CHN_AIN1, HAL_ADC_RESOLUTION_8);
    //将采集的电压量转化为光照强度值
    A0 = (float)((1 - adcValue/256.0) * 3.3*1000) - 1680;

    return A0;
}

```

(4) 在 `UseApp.c` 文件中实现解析控制命令函数 `usr_process_command_call()`，当上层应用发送控制命令时，解析命令的代码实现如下。

//解析收到的控制命令

```

int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {                               //关闭主动上报
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {                               //开启主动上报
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {                                //查询是否开启了主动上报功能
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);                     //命令数据
        }
    }
    if (0 == strcmp("A0", ptag)) {                                //查询光照强度值
        if (0 == strcmp("?", pval)) {
            updateA0();                                           //更新光照量数值
            ret = sprintf(pout, "A0=%.1f", A0);                   //命令数据
        }
    }
    if (0 == strcmp("V0", ptag)) {                                //查询主动上报的时间间隔
        if (0 == strcmp("?", pval)) {

```



```
        ret = sprintf(pout, "V0=%u", V0);           //命令数据
    }else{
        updateV0(pval);                             //更新主动上报的时间间隔
    }
}
return ret;                                         //返回命令数据
}
```

上述代码中调用了函数 `updateV0()` 来更新主动上报的时间间隔，具体的代码实现如下。

```
//更新主动上报时间间隔
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}
```

至此，光敏传感器节点的底层开发就完成了。由于不同传感器的参数标识和类型是不同的，初始化传感器的过程也不同，并且不同传感器采集数据的方式不同，所以当需要开发其他采集类的传感器时，只需要修改 `UseApp.h` 文件中的宏定义和 `UseApp.c` 文件中的函数 `sensor_init()` 和函数 `updateA0()` 即可。

2. 报警类传感器

人体红外传感器主要用于监测人的活动，当监测到活动人对象时，每隔 3 s 实时上报报警值 1，当人离开后，每隔 120 s 上报解除报警值 0，智云通信协议定义如表 2.7 所示，人体红外传感器程序逻辑驱动开发设计如图 2.31 所示。

表 2.7 传感器智云通信协议定义

传 感 器	属 性	参 数	权限	说 明
可燃气体	数值	A0	R	人体红外报警状态， 0 或 1
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态

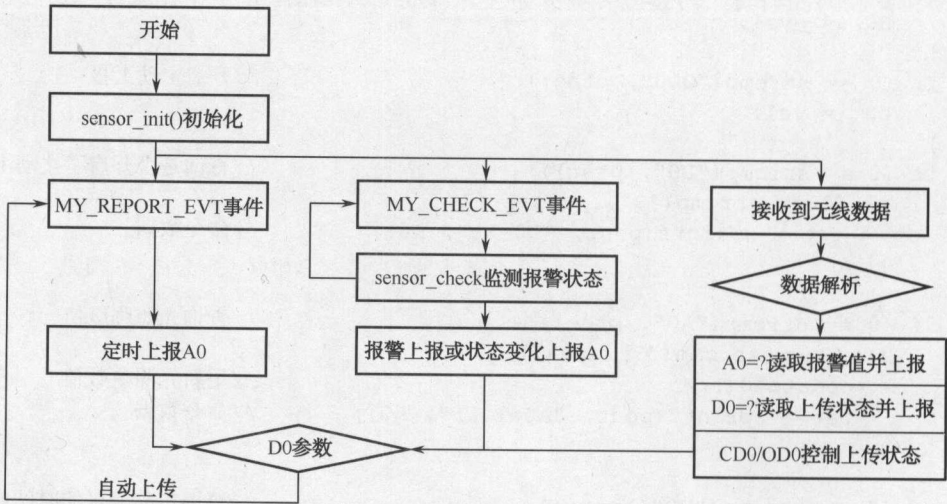


图 2.31 人体红外传感器监测程序逻辑



程序实现过程如下。

(1) 在 UserApp.h 文件中编写以下代码。

```
#define MY_REPORT_EVT 0x0001           //定义事件 MY_REPORT_EVT
#define MY_CHECK_EVT 0x0002           //定义事件 MY_CHECK_EVT
#define NODE_NAME "004"               //定义传感器参数标识
#define NODE_CATEGORY 1               //定义传感器类型
```

(2) 在 UseApp.c 文件中实现人体红外传感器的初始化 sensor_init(): 初始化传感器最基本的是配置选择寄存器和方向寄存器, 还要分别启动定时器来触发事件 MY_REPORT_EVT 和事件 MY_CHECK_EVT, 具体的代码实现如下。

```
//初始化传感器
void sensor_init(void)
{
    //配置 P0_5 端口为通用输入 IO
    SENSOR_SEL &= ~(SENSOR_BIT);
    SENSOR_DIR &= ~(SENSOR_BIT);

    //启动定时器, 触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10) *
        1000));
    //启动定时器, 触发事件: MY_CHECK_EVT
    osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, (uint16)((osal_rand()%10) *
        1000));
}
```

(3) 在 UseApp.c 文件中实现自定义事件处理函数 MyEventProcess(event), 代码实现如下。

```
//自定义事件处理
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        //主动上报报警状态值函数
        sensor_update();
        //启动定时器, 触发事件: MY_REPORT_EVT, 定时上报数据
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval *
            1000));
    }
    if (event & MY_CHECK_EVT) {
        //检测报警值函数
        sensor_check();
        //启动定时器, 触发事件: MY_CHECK_EVT, 定时查询报警值
        osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, 1000);
    }
}
```

其中, 事件 MY_REPORT_EVT 调用了函数 sensor_update() 来上报报警状态值, 具体的代码实现如下。

```
//主动上报报警状态值
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
```




```

uint8 *p = pData + 1;
int len;

//根据 D0 的位状态判定需要主动上报的数值
if ((D0 & 0x01) == 0x01){ //若报警值上报允许,则 pData 的数据包中添加报警值数据
    updateA0(); //更新警报状态值
    len = sprintf((char*)p, "A0=%u", A0);
    p += len;
    *p++ = ',';
}

//将需要上传的数据进行打包操作,并通过 zb_SendDataRequest() 发送到协调器
if (p - pData > 1) {
    pData[0] = '{';
    p[0] = 0;
    p[-1] = '}';

    zb_SendDataRequest(0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                        AF_DEFAULT_RADIUS );
    HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
}
}

```

事件 MY_CHECK_EVT 调用了函数 sensor_check()来检测报警状态值,具体的代码实现如下。

```

//检测报警值并决定是否报警
void sensor_check(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    int len;
    uint8 lastA0 = 0;

    if((D0 & 0x01) == 1){ //判断是否开启了主动上报
        lastA0 = A0; //记录上次 A0 的值
        updateA0(); //更新 A0 的值

        //当监测到维持高电平状态,上报报警值 A0=1
        if (A0 == 1) {
            if(Flag % 3 == 0){ //每 3 s 报警一次
                len = sprintf((char*)pData, "{A0=%u}", A0);
                //发送数据到协调器
                zb_SendDataRequest(0, cmd, len, (uint8*)pData, 0, AF_ACK_REQUEST,
                                    AF_DEFAULT_RADIUS);
                HalLedSet(HAL_LED_1, HAL_LED_MODE_BLINK); //通信 LED 闪烁一次
            }
            Flag++;
        }
        //当监测到维持低电平状态,上报清除报警状态 A0=0
        else if ((Flag != 0) && (lastA0 == 0) && (A0 == 0)) {
            len = sprintf((char*)pData, "{A0=%u}", A0);

```



```

//发送数据到协调器
zb_SendDataRequest(0, cmd, len, (uint8*)pData, 0, AF_ACK_REQUEST,
                   AF_DEFAULT_RADIUS);
HalLedSet(HAL_LED_1, HAL_LED_MODE_BLINK);           //通信 LED 闪烁一次
Flag = 0;
}
}
}

```

函数 `sensor_check()` 中调用了函数 `updateA0()` 来更新警报状态值，具体的代码实现如下。

```

uint8 updateA0(void)
{
    A0 = SENSOR_PIN;                                //根据 P0_5 口电平的高低来判断警报状态值
    return A0;
}

```

(4) 在 `UseApp.c` 文件中实现解析控制命令函数 `usr_process_command_call()`，当上层应用发送控制命令时，解析命令的代码实现如下。

```

//解析收到的控制命令
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {                    //关闭主动上报
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {                    //开启主动上报
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {                    //查询是否开启了主动上报
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);          //命令数据
        }
    }
    if (0 == strcmp("A0", ptag)) {                    //查询警报状态值
        if (0 == strcmp("?", pval)) {
            updateA0();                                //更新警报状态值
            ret = sprintf(pout, "A0=%u", A0);          //命令数据
        }
    }
    if (0 == strcmp("V0", ptag)) {
        if (0 == strcmp("?", pval)) {                //查询主动上报时间间隔
            ret = sprintf(pout, "V0=%u", V0);        //命令数据
        } else {
            updateV0(pval);                          //更新主动上报时间间隔
        }
    }
}

```



```
    }  
    return ret;                                     //返回命令数据  
}
```

上述代码中还调用了函数 `updataV0()`来更新主动上报的时间间隔，具体的代码实现如下。

```
//更新主动上报时间间隔  
uint16 updateV0(char *val)  
{  
    //将字符串变量 val 解析转换为整型变量赋值  
    myReportInterval = atoi(val);  
    V0 = myReportInterval;  
    return V0;  
}
```

至此，人体红外传感器节点的底层开发就完成了，由于不同传感器的参数标识和类型是不同的，初始化传感器的过程也不同，警报状态值与不同的 IO 口的电平高低有关，所以当需要开发其他报警类的传感器时，只需要修改 `UseApp.h` 文件中的宏定义和 `UseApp.c` 文件中的函数 `sensor_init()`和函数 `updataA0()`即可。

3. 控制类传感器

继电器属于典型的控制类设备，可通过发送执行命令控制继电器的开关，ZXBee 协议定义如表 2.8 所示，继电器程序逻辑如图 2.32 所示。

表 2.8 继电器智云通信协议定义

传 感 器	属 性	参 数	权限	说 明
继电器	继电器开关	D1(OD1/CD1)	R(W)	D1 的 Bit 表示各路继电器开合状态，OD1 为开、CD1 为合

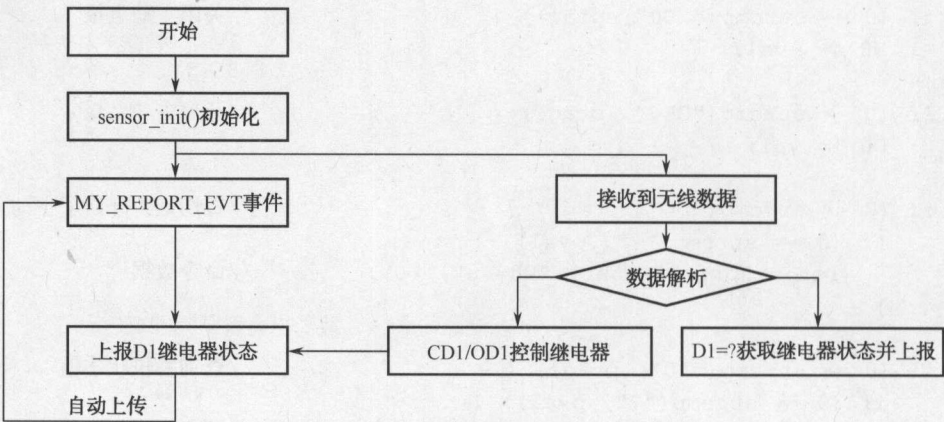


图 2.32 继电器程序逻辑

程序实现过程如下。

(1) 在 `UserApp.h` 文件中编写以下代码。

```
#define MY_REPORT_EVT 0x0001                                     //定义事件 MY_REPORT_EVT  
#define NODE_NAME "003"                                         //定义传感器参数标识  
#define NODE_CATEGORY 1                                         //定义传感器类型
```

(2)在 `UseApp.c` 文件中实现继电器传感器的初始化 `sensor_init()`：初始化最基本的是配置，选择寄存器和方向寄存器，启动一个定时器来触发事件 `MY_REPORT_EVT`，具体的代码实现



如下。

```
//初始化传感器
void sensor_init(void)
{
    //配置 P0_1、P0_5 端口为通用输出 IO
    SENSOR_SEL &= ~(SENSOR_BIT);
    SENSOR_DIR |= SENSOR_BIT;
    SENSOR_PORT |= SENSOR_BIT;           //LS1/LS2 断开

    //启动定时器，触发事件：MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10) *
        1000));
}
```

(3)在 UseApp.c 文件中实现自定义事件处理函数 MyEventProcess(event), 代码实现如下。

```
//自定义事件处理
void MyEventProcess( uint16 event )
{
    if(event & MY_REPORT_EVT) {
        //主动上报数据
        sensor_update();
        //启动定时器，触发事件：MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID,MY_REPORT_EVT, (uint16)(myReportInterval *
            1000));
    }
}
```

上述代码中调用了函数 sensor_update()来上报采集到的数据，具体的代码实现如下。

```
//处理主动上报的数据
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){ //若控制编码上报允许，则 pData 的数据包中添加控制编码数据
        len = sprintf((char*)p, "D1=%u", D1);
        p += len;
        *p++ = ',';
    }

    //将需要上传的数据进行打包操作，并通过 zb_SendDataRequest() 发送到协调器
    if (p - pData > 1) {
        pData[0] = '{';
        p[0] = 0;
        p[-1] = '}';

        zb_SendDataRequest(0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
            AF_DEFAULT_RADIUS );
    }
```



```

        HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK );    //通信 LED 闪烁一次
    }
}

```

(4) 在 UseApp.c 文件中实现解析控制命令函数 `usr_process_command_call()`，当上层应用发送控制命令时，解析命令的代码实现如下。

```

//解析收到的控制命令
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {                //关闭主动上报
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {                //开启主动上报
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {                //查询是否开启了主动上报功能
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);    //命令数据
        }
    }
    if (0 == strcmp("CD1", ptag)) {                //关闭继电器命令
        D1 &= ~val;
        sensor_control(D1);                    //调用函数来关闭继电器
    }
    if (0 == strcmp("OD1", ptag)) {                //打开继电器命令
        D1 |= val;
        sensor_control(D1);                    //调用函数来打开继电器
    }
    if (0 == strcmp("D1", ptag)) {                //查询继电器状态
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D1=%u", D1);    //命令数据
        }
    }
    if (0 == strcmp("V0", ptag)) {                //查询主动上报的时间间隔
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "V0=%u", V0);    //命令数据
        } else {
            updateV0(pval);                    //更新主动上报的时间间隔
        }
    }
    return ret;                                //返回命令数据
}

```

上述代码中调用了函数 `sensor_control()` 来控制继电器的状态，具体的代码实现如下。



```
//控制继电器的状态
void sensor_control(uint8 cmd)
{
    if (cmd == 0){
        SENSOR_PORT |= 0x22;                //LS1/LS2 断开
    } else if (cmd == 1){
        SENSOR_PORT &= ~0x02;                //LS1 闭合
        SENSOR_PORT |= 0x20;                //LS2 断开
    } else if (cmd == 2){
        SENSOR_PORT |= 0x02;                //LS1 断开
        SENSOR_PORT &= ~0x20;                //LS2 闭合
    } else if (cmd == 3){
        SENSOR_PORT &= ~0x22;                //LS1/LS2 闭合
    }
}
```

另外，还调用了函数 `updateV0()` 来更新主动上报的时间间隔，具体的代码实现如下。

```
//更新主动上报时间间隔
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}
```

至此，继电器传感器节点的底层开发就完成了。由于不同传感器的参数标识和类型不同，初始化传感器的过程也不同，还有控制传感器状态的方式不同，所以当需要开发其他控制类的传感器时，只需要修改 `UseApp.h` 文件中的宏定义和 `UseApp.c` 文件中的函数 `sensor_init()` 和函数 `sensor_control()` 即可。

2.3.5 开发步骤

1. 智云硬件环境搭建

- (1) 一台 S210 系列 Android 开发平台，1 个温湿度传感器无线节点。
- (2) 安装 ZStack 的安装包，在开发资源包“03-ZigBee 安装资料\ZStack\ZStack-CC2530-2.4.0-1.4.0.exe”，安装完后默认生成“C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0”文件夹。
- (3) 打开例程：将“04-开发例程\Chapter02 智云物联开发基础\任务 05-智云硬件驱动开发\ZXBee CC2530”目录下所有文件夹复制到“C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples”文件夹下。
- (4) 分别打开协调器和传感器工程，编译代码。
- (5) 把 CC2530 仿真器连接到 CC2530 无线节点，使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中。

(6) 组成智云无线传感网络，将数据接入到智云服务中心。

2. 以光敏节点、人体红外节点和继电器节点为例

- (1) 按照上述的环境搭建，将光敏节点、人体红外节点、继电器节点、协调器节点编译



后烧写到对应的传感器。

(2) 准备一台智云 Android 开发平台。

(3) 组成智云无线传感网络，并将数据接入到智云服务中心。

(4) 参考 2.1 节内容和步骤对智云 Android 开发平台进行配置，确保智云网络连接成功。

(5) 运行 ZCloudTools 工具对节点进行调试。部分测试步骤（以继电器为例）如下。

单击“综合演示”图标，进入节点拓扑图综合演示界面，等待一段时间后，就会形成所有传感节点的拓扑图结构，包括协调器（红色）、路由节点（紫色）和终端节点（浅蓝色），如图 2.33 所示。

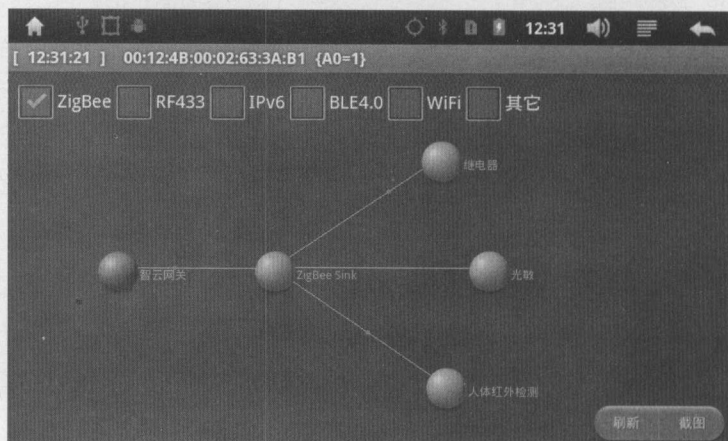


图 2.33 节点拓扑图

单击继电器节点图标，进入继电器节点控制界面。单击开关，控制继电器开合，进而控制灯光亮灭，如图 2.34 所示。



图 3.34 继电器节点控制

返回主界面，单击“数据分析”图标，进入数据分析界面。

单击节点列表中的“继电器”，进入继电器节点调试界面。输入调试指令“{D1=?}”并发送，查询当前继电器状态值，如图 2.35 所示。

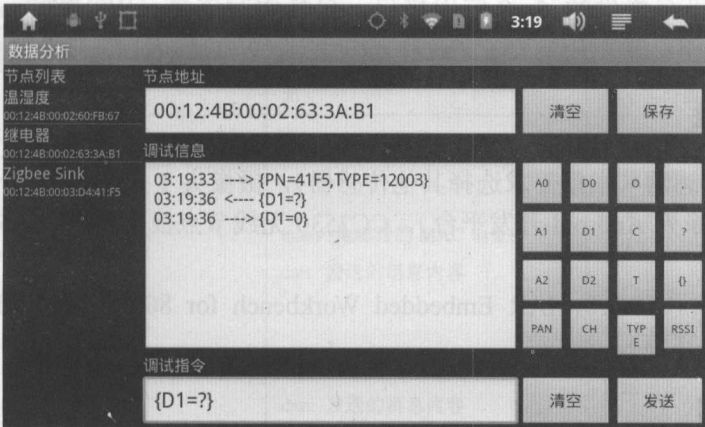


图 2.35 查询继电器状态

输入调试指令“{CD0=1}”并发送，修改继电器状态值为 1（即“开”状态）并查询当前继电器状态值，指令成功执行后会听到继电器开合的声音，执行结果如图 2.36 所示。

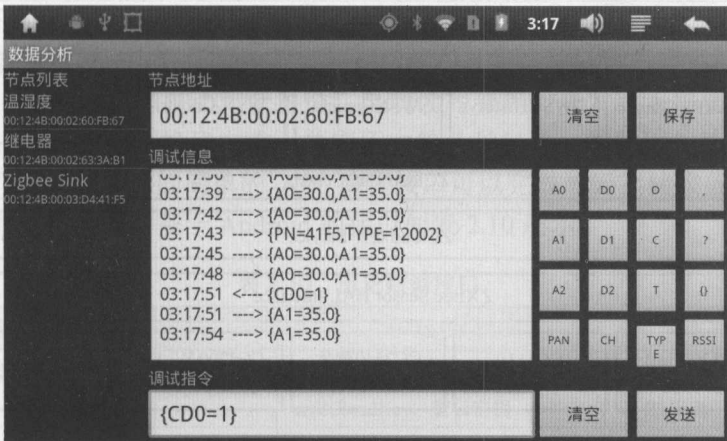


图 2.36 修改继电器状态

2.3.6 总结与拓展

无线节点的传感器代码见本任务的目录“04-开发例程\Chapter02 智云物联开发基础\任务 05-智云硬件驱动开发\ZXBee CC2530”，可根据需求修改可燃气体传感器代码为报警类传感器代码。

2.4 任务 8：AndroidAPI 开发

2.4.1 学习目标

- 理解 ZigBee 智云通信协议程序逻辑；
- 掌握传感器节点的硬件驱动开发；



- 掌握智云平台提供的 5 个应用接口，包括实时连接（WSNRTConnect）、历史数据（WSNHistory）、摄像头（WSNCamera）、自动控制（WSNAutoctrl）、用户数据（WSNProperty）。

2.4.2 开发环境

硬件：温度传感器（根据需求选择其他传感器），摄像头 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 2 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

2.4.3 原理学习

智云物联云平台提供五大应用接口供开发者使用，包括实时连接（WSNRTConnect）、历史数据（WSNHistory）、摄像头（WSNCamera）、自动控制（WSNAutoctrl）、用户数据（WSNProperty），详细接口架构图如图 2.37 所示。

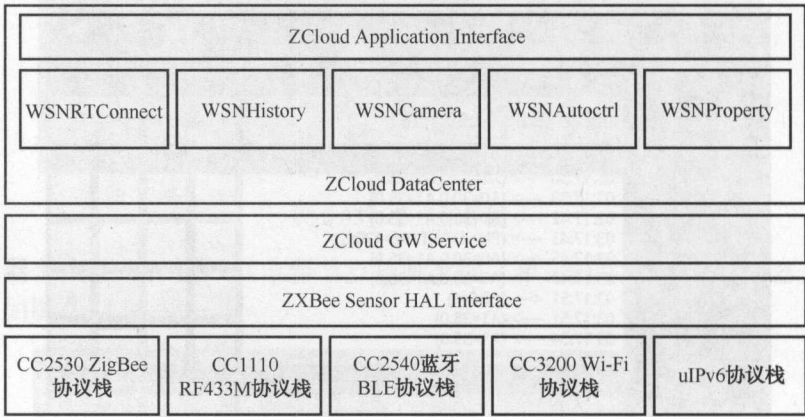


图 2.37 接口架构图

针对 Android 移动应用程序开发，智云平台提供应用接口库 libwsnDroid2.jar，开发者只需要在编写 Android 应用程序时，先导入该 jar 包，然后在代码中调用相应的方法即可。

1. 实时连接接口

实时连接接口基于智云平台的消息推送服务，消息推送服务通过利用云端与客户端之间建立稳定、可靠的长连接来为开发者提供向客户端应用推送实时消息服务。智云消息推送服务针对物联网行业特征，支持多种推送类型，如传感实时数据、执行控制命令、地理位置信息、SMS 短信消息等，同时提供用户信息及通知消息统计信息，方便开发者进行后续开发及运营。基于 Android 的接口如表 2.9 所示。

表 2.9 实时连接接口

函 数	参 数 说 明	功 能
new WSNRTConnect(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	建立实时数据实例，并初始化智云 ID 及密钥



续表

函 数	参 数 说 明	功 能
connect()	无	建立实时数据服务连接
disconnect()	无	断开实时数据服务连接
setRTConnectListener(){ onConnect() onConnectLost(Throwable arg0) onMessageArrive(String mac, byte[] dat) }	mac: 传感器的 MAC 地址; dat: 发送的消息内容	设置监听, 接收实时数据服务推送过来的消息; onConnect: 连接成功操作; onConnectLost: 连接失败操作; onMessageArrive: 数据接收操作
sendMessage(String mac, byte[] dat)	mac: 传感器的 MAC 地址; dat: 发送的消息内容	发送消息
setServerAddr(String sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥 (需要重新断开连接)

2. 历史数据接口

历史数据基于智云数据中心提供的智云数据库接口开发, 智云数据库采用 Hadoop 后端分布式数据库集群, 并且多机房自动冗余备份, 自动读写分离, 开发者不需要关注后端机器及数据库的稳定性、网络问题、机房灾难、单库压力等各种风险。物联网传感器数据可以在智云数据库永久保存, 通过提供的简单的 API 编程接口可以完成与云存储服务器的数据连接、数据访问存储、数据使用等, 基于 Android 的接口如表 2.10 所示。

表 2.10 历史数据接口

函 数	参 数 说 明	功 能
new WSNHistory(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化历史数据对象, 并初始化智云 ID 及密钥
queryLast1H(String channel);	channel: 传感器数据通道	查询最近 1 小时的历史数据
queryLast6H(String channel);	channel: 传感器数据通道	查询最近 6 小时的历史数据
queryLast12H(String channel);	channel: 传感器数据通道	查询最近 12 小时的历史数据
queryLast1D(String channel);	channel: 传感器数据通道	查询最近 1 天的历史数据
queryLast5D(String channel);	channel: 传感器数据通道	查询最近 5 天的历史数据
queryLast14D(String channel);	channel: 传感器数据通道	查询最近 14 天的历史数据
queryLast1M(String channel);	channel: 传感器数据通道	查询最近 1 月 (30 天) 的历史数据
queryLast3M(String channel);	channel: 传感器数据通道	查询最近 3 月 (90 天) 的历史数据
queryLast6M(String channel);	channel: 传感器数据通道	查询最近 6 月 (180 天) 的历史数据
queryLast1Y(String channel);	channel: 传感器数据通道	查询最近 1 年 (365 天) 的历史数据



续表

函 数	参 数 说 明	功 能
query();	无	获取所有通道最后一次数据
query(String channel);	channel: 传感器数据通道	获取该通道下最后一次数据
query(String channel, String start, String end);	channel: 传感器数据通道; start: 起始时间; end: 结束时间; 时间为 ISO 8601 格式的日期, 如 2010-05-20T11:00:00Z	通过起止时间查询指定时间段的历史数据
query(String channel, String start, String end, String interval);	channel: 传感器数据通道; start: 起始时间; end: 结束时间; interval: 采样点的时间间隔, 详见后续说明。 时间为 ISO 8601 格式的日期, 如 2010-05-20T11:00:00Z	通过起止时间查询指定时间段指定时间间隔的历史数据
setServerAddr(String sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥

说明:

(1) 每次采样的数据点最大个数为 1500。

(2) 历史数据返回格式示例 (压缩的 JSON 格式):

```
{ "current_value": "11.0", "datapoints": [ { "at": "2015-08-30T14:30:14Z", "value": "11.0" }, { "at": "2015-08-30T14:30:24Z", "value": "11.0" }, { "at": "2015-08-30T14:30:34Z", "value": "12.0" }, ..... { "at": "2015-08-30T15:29:54Z", "value": "11.0" }, { "at": "2015-08-30T15:30:04Z", "value": "11.0" } ], "id": "00:12:4B:00:02:37:7E:7A_A0", "at": "2015-08-30T15:30:04Z" }
```

(3) 历史数据接口支持动态的调整采样间隔, 当查询函数没有赋值 interval 参数时, 采样间隔遵循以下原则取点如表 2.11 所示。

表 2.11 时间采样

一次查询支持的最大查询范围	interval 默认取值	描 述
≤ 6 hours	0	提取存储的每个点
≤ 12 hours	30	每 30 秒取一个点
≤ 24 hours	60	每 1 分钟取一个点
≤ 5 days	300	每 5 分钟取一个点
≤ 14 days	900	每 15 分钟取一个点
≤ 30 days	1800	每 30 分钟取一个点
≤ 90 days	10800	每 3 小时取一个点
≤ 180 days	21600	每 6 小时取一个点
≤ 365 days	43200	每 12 小时取一个点
> 365 days	86400	每 24 小时取一个点



3. 摄像头接口

智云平台提供对 IP 摄像头的远程采集控制接口，支持远程对视频图像进行实时采集、图像抓拍、控制云台转动等操作。基于 Android 的接口如表 2.12 所示。

表 2.12 历史数据接口

函 数	参 数 说 明	功 能
new WSNCamera(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化摄像头对象, 并初始化智云 ID 及密钥
initCamera(String myCameraIP, String user, String pwd, String type);	myCameraIP: 摄像头外网域名/IP 地址; user: 摄像头用户名; pwd: 摄像头密码; type: 摄像头类型 (F-Series、F3-Series、H3-Series); # 以上参数从摄像头手册获取	设置摄像头域名、用户名、密码、类型等参数
openVideo();	无	打开摄像头
closeVideo();	无	关闭摄像头
control(String cmd);	cmd: 云台控制命令, 参数如下。 UP: 向上移动一次; DOWN: 向下移动一次; LEFT: 向左移动一次; RIGHT: 向右移动一次; HPATROL: 水平巡航转动; VPATROL: 垂直巡航转动; 360PATROL: 360° 巡航转动	发指令控制摄像头云台转动
checkOnline();	无	检测摄像头是否在线
snapshot();	无	抓拍照片
setCameraListener(){ onOnline(String myCameraIP, boolean online) onSnapshot(String myCameraIP, Bitmap bmp) onVideoCallBack(String myCameraIP, Bitmap bmp) }	myCameraIP: 摄像头外网域名/IP 地址; online: 摄像头在线状态 (0/1); bmp: 图片资源	监听摄像头返回数据: onOnline: 摄像头在线状态返回; onSnapshot: 返回摄像头截图; onVideoCallBack: 返回实时的摄像头视频图像
freeCamera(String myCameraIP);	myCameraIP: 摄像头外网域名/IP 地址	释放摄像头资源
setServerAddr(String sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥

4. 自动控制接口

智云物联平台提供了一个操作简单但是功能强大的逻辑编辑器，为开发物联网系统编辑复杂的控制逻辑，可以实现数据更新、设备状态查询、定时硬件系统控制、定时发送短消息、根



据各种变量触发某个复杂控制策略实现系统复杂控制等。智云自动控制接口基于触发逻辑单元的自动控制功能，触发器、执行器、执行策略、执行记录保存在智云数据中心，设计步骤如下。

- 为每个传感器、执行器的关键数据和控制量创建一个变量；
- 新建基础控制策略，控制策略里可以运用上一步新建的变量；
- 新建复杂控制策略，复杂控制策略可以使用运算符，可以无穷组合基础控制策略。

基于 Android 的接口如表 2.13 所示。

表 2.13 自动控制接口

函 数	参 数 说 明	功 能
new WSNAutoctrl(String myZCloudID,String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化自动控制对象, 并初始化智云 ID 及密钥
createTrigger(String name, String type, JSONObject param);	name: 触发器名称; type: 触发器类型 (sensor、timer); param: 触发器内容, JSON 对象格式, 创建成功后返回该触发器 ID (JSON 格式)	创建触发器
createActuator(String name,String type,JSONObject param);	name: 执行器名称; type: 执行器类型 (sensor、ipcamera、phone、job); param: 执行器内容, JSON 对象格式, 创建成功后返回该执行器 ID (JSON 格式)	创建执行器
createJob(String name, boolean enable, JSONObject param);	name: 任务名称; enable: true (使能任务)、false (禁止任务); param: 任务内容, JSON 对象格式, 创建成功后返回该任务 ID (JSON 格式)	创建任务
deleteTrigger(String id);	id: 触发器 ID	删除触发器
deleteActuator(String id);	id: 执行器 ID	删除执行器
deleteJob(String id);	id: 任务 ID	删除任务
setJob(String id,boolean enable);	id: 任务 ID; enable: true (使能任务)、false (禁止任务)	设置任务使能开关
deleteSchedudler(String id);	id: 任务记录 ID	删除任务记录
getTrigger();	无	查询当前智云 ID 下的所有触发器内容
getTrigger(String id);	id: 触发器 ID	查询该触发器 id 内容
getTrigger(String type);	type: 触发器类型	查询当前智云 ID 下的所有该类型的触发器内容
getActuator();	无	查询当前智云 ID 下的所有执行器内容
getActuator(String id);	id: 执行器 ID	查询该执行器 ID 内容
getActuator(String type);	type: 执行器类型	查询当前智云 ID 下的所有该类型的执行器内容
getJob();	无	查询当前智云 ID 下的所有任务内容

续表

函 数	参 数 说 明	功 能
getJob(String id);	id: 任务 ID	查询该任务 ID 内容
getSchedudler();	无	查询当前智云 ID 下的所有任务记录内容
getSchedudler(String jid,String duration);	id: 任务记录 ID duration:duration=×<year month day hours minute> //默认返回 1 天的记录	查询该任务记录 ID 某个时间段的内容
setServerAddr(String sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥

5. 用户数据接口

智云用户数据接口提供私有的数据库使用权限，实现多客户端间共享的私有数据进行存储、查询和使用。私有数据存储采用 key-value 型数据库服务，编程接口更简单高效，基于 Android 的接口如表 2.14 所示。

表 2.14 历史数据接口

函 数	参 数 说 明	功 能
new WSNProperty(String myZCloudID,String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化用户数据对象,并初始化智云 ID 及密钥
put(String key,String value);	key: 名称; value: 内容	创建用户应用数据
get();	无	获取所有的键值对
get(String key);	key: 名称	获取指定 key 的 value 值
setServerAddr(String sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(String myZCloudID, String myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥

2.4.4 开发内容

结合智云节点和 ZXBee 协议，开发了一套基于 Android 的简单的 libwsnDroidDemo 程序（该程序在开发资源包的“04-开发例程\chapter02-智云物联开发基础\任务 06-智云 Android 简单应用”目录下）。根据 2.4.3 中实现的接口，该应用的功能主要是传感器的读取与控制、历史数据查询与曲线显示、摄像头的控制、自动控制和应用数据存储与读取。

为了让程序更有可读性，该应用使用 2 个包，每个包分为多个 Activity 类，使用接口实现控制与数据的存取，其中，在 com.zhiyun360.wsn.auto 包下是对自动控制接口中的方法进行调用与实现的，主 Activity 只需要实现通过单击不同的按钮跳转到其他 Activity 中，在 src 包中的目录结构如图 2.38 所示。



图 2.38 src 目录结构

其中，DemoActivity 即为主 Activity，作为一个引导作用，用来链接到其他 Activity，也可在 DemoActivity.java 文件中定义静态变量，每个 Activity 都应有自己的布局。

1. 实时连接接口

要实现传感器实时数据的发送需要在 SensorActivity.java 文件中调用类 WSNRTConnect 的几个方法即可，具体调用方法及步骤如下。

(1) 连接服务器地址。外网服务器地址及端口默认为 zhiyun360.com:28081，如果用户需要修改，调用方法 setServerAddr(sa)进行设置即可。

```
wRTConnect.setServerAddr(zhiyun360.com:28081); //设置外网服务器地址及端口
```

(2) 初始化智云 ID 及密钥。先定义序列号和密钥（序列号和密钥为用户注册云平台账户时所需的传感器序列号和密钥），然后初始化，本示例中是在 DemoActivity 中设置 ID 与 Key，并在每个 Activity 中直接调用即可，后续不再赘述。

```
String myZCloudID = "12345678"; //序列号
String myZCloudKey = "12345678"; //密钥
wRTConnect = new WSNRTConnect(DemoActivity.myZCloudID,DemoActivity.myZCloudKey);
```

(3) 建立数据推送服务连接。

```
wRTConnect.connect(); //调用 connect 方法
```

(4) 注册数据推送服务监听器，接收实时数据服务推送过来的消息。

```
wRTConnect.setRTConnectListener(new WSNRTConnectListener() {
    @Override
    public void onConnect() { //连接服务器成功
        //TODO Auto-generated method stub
    }

    @Override
    public void onConnectLost(Throwable arg0) { //连接服务器失败
        //TODO Auto-generated method stub
    }

    @Override
    public void onMessageArrive(String arg0, byte[] arg1) { //数据到达
        //TODO Auto-generated method stub
    }
});
```




```
    }
    });
```

(5) 实现消息发送。调用 `sendMessage` 方法向指定的传感器发送消息。

```
String mac = "00:12:4B:00:03:A7:E1:17";           //目的地址
String dat = "{OD1=1,D1=?}"                       //数据指令格式
wRTConnect.sendMessage(mac, dat.getBytes());       //发送消息
```

(6) 断开数据推送服务。

```
wRTConnect.disconnect();
```

(7) `SensorActivity` 的完整示例。下面是一个完整的 `SensorActivity.java` 代码示例，源码参考 `libwsnDroidDemo/src/SensorActivity.java`。

```
public class SensorActivity extends Activity {
    private Button mBTNOpen,mBTNClose;
    private TextView mTVInfo;
    private WSNRTConnect wRTConnect;

    private void textInfo(String s) {
        mTVInfo.setText(mTVInfo.getText().toString() + "\n" + s);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.sensor);
        setTitle("传感器数据采集与控制模块");
        mBTNOpen = (Button) findViewById(R.id.btnOpen);
        mBTNClose = (Button) findViewById(R.id.btnClose);

        mTVInfo = (TextView) findViewById(R.id.tvInfo);
        //实例化 WSNRTConnect, 并初始化智云 ID 和 KEY
        wRTConnect =
            new WSNRTConnect(DemoActivity.myZCloudID,DemoActivity.myZCloudKey);
        //设置 WSNRTConnect 服务器地址
        wRTConnect.setServerAddr("zhiyun360.com:28081");
        //设置监听器
        mBTNClose.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //TODO Auto-generated method stub
                String mac = "00:12:4B:00:03:A7:E1:17";
                String dat = "{CD1=1,D1=?}";
                textInfo(mac + " <<< " + dat);
                wRTConnect.sendMessage(mac, dat.getBytes());
            }
        });
        //建立连接
        wRTConnect.connect();

        mBTNOpen.setOnClickListener(new OnClickListener() {
            @Override
```



```
        public void onClick(View arg0) {
            //TODO Auto-generated method stub
            String mac = "00:12:4B:00:03:A7:E1:17";
            String dat = "{OD1=1,D1=?}";
            textInfo(mac + " <<< " + dat);
            WRTConnect.sendMessage(mac, dat.getBytes());
        }
    });

    WRTConnect.setRTConnectListener(new WSNRTConnectListener() {
        @Override
        public void onConnect() {
            //TODO Auto-generated method stub
            textInfo("connected to server");
        }

        @Override
        public void onConnectLost(Throwable arg0) {
            //TODO Auto-generated method stub
            textInfo("connection lost");
        }

        @Override
        public void onMessageArrive(String arg0, byte[] arg1) {
            //TODO Auto-generated method stub
            textInfo(arg0 + " >>> " + new String(arg1));
        }
    });

    textInfo("connecting...");
}
@Override
public void onDestroy() {
    WRTConnect.disconnect(); //断开连接
    super.onDestroy();
}
}
```

2. 历史数据接口

同理,要实现获取传感器的历史数据需要在 HistoryActivity.java 文件中调用类 WSNHistory 的几个方法即可,具体调用方法及步骤如下。

(1) 实例化历史数据对象,直接实例化并连接。

(2) 连接服务器地址,外网服务器地址及端口默认为 zhiyun360.com:28081,如果需要修改,调用 setServerAddr(sa) 方法进行设置即可。

```
WRTConnect.setServerAddr(zhiyun360.com:28081); //设置外网服务器地址及端口
```

(3) 初始化智云 ID 及密钥。先定义序列号和密钥,然后初始化。

```
String myZCloudID = "12345678"; //序列号
String myZCloudKey = "12345678"; //密钥
//初始化智云 ID 及密钥
```



```
wHistory = newWSNHistory (DemoActivity.myZCloudID,DemoActivity.myZCloudKey);
```

(4) 查询历史数据。以下方法为查询自定义时段的历史数据，如需要查询其他时间段（如最近1个小时、最近一个月）历史数据，请参考2.4.3节API的介绍。

```
wHistory.queryLast1H(String channel);
```

```
wHistory.queryLast1M(String channel);
```

(5) HistoryActivity 的完整示例。下面是一个完整的 HistoryActivity.java 代码示例，源码参考 SDK 包/Android/libwsnDroidDemo/src/HistoryActivity.java。

```
public class HistoryActivity extends Activity implements OnClickListener {
    private String channel = "00:12:4B:00:02:CB:A8:52_A0"; //定义数据流通道
    Button mBTN1H, mBTN6H, mBTN12H, mBTN1D, mBTN5D, mBTN14D, mBTN1M, mBTN3M,
    mBTN6M, mBTN1Y, mBTNSTART, mBTNEND, mBTNQUERY;
    TextView mTVData;
    SimpleDateFormat simpleDateFormat;
    SimpleDateFormat outputDateFormat;
    WSNHistory wHistory; //定义历史数据对象
    @SuppressWarnings("SimpleDateFormat")
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.histroy);
        simpleDateFormat = new SimpleDateFormat("yyyy-M-d");
        outputDateFormat = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
        mTVData = (TextView) findViewById(R.id.tvData);
        mBTN1H = (Button) findViewById(R.id.btn1h);
        mBTN6H = (Button) findViewById(R.id.btn6h);
        mBTN12H = (Button) findViewById(R.id.btn12h);
        mBTN1D = (Button) findViewById(R.id.btn1d);
        mBTN5D = (Button) findViewById(R.id.btn5d);
        mBTN14D = (Button) findViewById(R.id.btn14d);
        mBTN1M = (Button) findViewById(R.id.btn1m);
        mBTN3M = (Button) findViewById(R.id.btn3m);
        mBTN6M = (Button) findViewById(R.id.btn6m);
        mBTN1Y = (Button) findViewById(R.id.btn1y);
        mBTNSTART = (Button) findViewById(R.id.btnStart);
        mBTNEND = (Button) findViewById(R.id.btnEnd);
        mBTNQUERY = (Button) findViewById(R.id.query);
        //为每个按钮设置监听器响应单击事件
        mBTN1H.setOnClickListener(this);
        mBTN6H.setOnClickListener(this);
        mBTN12H.setOnClickListener(this);
        mBTN1D.setOnClickListener(this);
        mBTN5D.setOnClickListener(this);
        mBTN14D.setOnClickListener(this);
        mBTN1M.setOnClickListener(this);
        mBTN3M.setOnClickListener(this);
        mBTN6M.setOnClickListener(this);
        mBTN1Y.setOnClickListener(this);
        mBTNSTART.setOnClickListener(this);
        mBTNEND.setOnClickListener(this);
    }
}
```




```
mBTNQUERY.setOnClickListener(this);
wHistory = new WSNHistory();           //初始化历史数据对象
//初始化智云 ID 和密钥
wHistory.initZCloud(DemoActivity.myZCloudID, DemoActivity.myZCloudKey);
}
//为按钮实现单击事件
@Override
public void onClick(View arg0) {
    //TODO Auto-generated method stub
    mTVData.setText("");
    String result = null;
    try {
        if (arg0 == mBTN1H) { //查询近 1 小时的历史数据
            result = wHistory.queryLast1H(channel);
        }
        if (arg0 == mBTN6H) { //查询近 6 小时的历史数据
            result = wHistory.queryLast6H(channel);
        }
        if (arg0 == mBTN12H) { //查询近 12 小时的历史数据
            result = wHistory.queryLast12H(channel);
        }
        if (arg0 == mBTN1D) { //查询近 1 天的历史数据
            result = wHistory.queryLast1D(channel);
        }
        if (arg0 == mBTN5D) { //查询近 5 天的历史数据
            result = wHistory.queryLast5D(channel);
        }
        if (arg0 == mBTN14D) { //查询近 14 天的历史数据
            result = wHistory.queryLast14D(channel);
        }
        if (arg0 == mBTN1M) { //查询近 1 个月的历史数据
            result = wHistory.queryLast1M(channel);
        }
        if (arg0 == mBTN3M) { //查询近 3 个月的历史数据
            result = wHistory.queryLast3M(channel);
        }
        if (arg0 == mBTN6M) { //查询近 6 个月的历史数据
            result = wHistory.queryLast6M(channel);
        }
        if (arg0 == mBTN1Y) { //查询近 1 年的历史数据
            result = wHistory.queryLast1Y(channel);
        }
        if (arg0 == mBTNSTART) { //设置要查询数据的起始时间
            new DatePickerDialog(this,
                new DatePickerDialog.OnDateSetListener() {
                    @Override
                    public void onDateSet(DatePicker view, int year,
                        int monthOfYear, int dayOfMonth) {
                        mBTNSTART.setText(year + "-"
                            + (monthOfYear + 1) + "-" + dayOfMonth);
                    }
                }
            ).show();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



```

    }
    }, 2014, 0, 1).show();
}
if (arg0 == mBTNEND) { //设置要查询数据的截止时间
    new DatePickerDialog(this,
        new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int year,
                int monthOfYear, int dayOfMonth) {
                mBTNEND.setText(year + "-" + (monthOfYear + 1)
                    + "-" + dayOfMonth);
            }
        }, 2014, 0, 1).show();
}
if (arg0 == mBTNQUERY) { //单击查询按钮
    Date sdate = simpleDateFormat.parse(mBTNSTART.getText().toString());
    Date edate = simpleDateFormat.parse(mBTNEND.getText().toString());
    String start = outputDateFormat.format(sdate) + "Z";
    String end = outputDateFormat.format(edate) + "Z";
    result = wHistory.queryLast(start, end, "0", channel); //调用查询函数
}
mTVData.setText(jsonFormatter(result)); //显示数据
} catch (Exception e) {
    e.printStackTrace();
    Toast.makeText(getApplicationContext(), "查询数据失败, 请重试!",
        Toast.LENGTH_SHORT).show();
}
}

public static String jsonFormatter(String uglyJSONString) {
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    JsonParser jp = new JsonParser();
    JsonElement je = jp.parse(uglyJSONString);
    String prettyJsonString = gson.toJson(je);
    return prettyJsonString;
}
}

```

(6) 本任务也实现了历史数据曲线显示, 在 `HistoryActivityEx.java` 类中, 调用同样的方法初始化并建立连接, 后引用 `java.text.SimpleDateFormat` 包中的方法进行 `data->text` 格式转换, 代码如下。

```

SimpleDateFormat outputDateFormat = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
JSONObject jsonObj = new JSONObject(result);
JSONArray datapoints = jsonObj.getJSONArray("datapoints");
if (datapoints.length() == 0) {
    Toast.makeText(getApplicationContext(), "获取数据点为 0!",
        Toast.LENGTH_SHORT).show();
    return;
}
for (int i = 0; i < datapoints.length(); i++) {

```



```
JSONObject jsonObj = datapoints.getJSONObject(i);

String val = jsonObj.getString("value");
String at = jsonObj.getString("at");

Double dval = Double.parseDouble(val);
Date dat = outputDateFormat.parse(at);

xlist.add(dat);
ylist.add(dval);
}
```

(7) 引用 `org.achartengine` 中的子类, 可以实现数据图表显示, 代码如下。

```
XYMultipleSeriesRenderer renderer = new XYMultipleSeriesRenderer();
renderer.setAxisTitleTextSize(16); // 数轴文字字体大小
renderer.setChartTitleTextSize(20); // 标题字体大小
renderer.setLabelsTextSize(15); // 数轴刻度字体大小
renderer.setLegendTextSize(15); // 曲线
renderer.setPointSize(5f);
renderer.setMargins(new int[] { 20, 30, 15, 20 });

XYSeriesRenderer r = new XYSeriesRenderer();
r.setColor(Color.rgb(30, 144, 255));
//r.setPointStyle(PointStyle.CIRCLE);
r.setFillPoints(false);
r.setLineWidth(1);
r.setDisplayChartValues(true);
renderer.addSeriesRenderer(r); // 加载曲线信息

renderer.setApplyBackgroundColor(true);
renderer.setBackgroundColor(Color.WHITE);
renderer.setXLabels(10);
renderer.setYLabels(10);
renderer.setShowGrid(true);
renderer.setMarginsColor(Color.WHITE);
renderer.setZoomButtonsVisible(true);

renderer.setChartTitle("");
renderer.setXTitle("时间");
renderer.setYTitle("数值");
renderer.setXAxisMin(xlist.get(0).getTime());
renderer.setXAxisMax(xlist.get(xlist.size()-1).getTime());
renderer.setYAxisMin(minValue);
renderer.setYAxisMax(maxValue); // 数轴上限
renderer.setAxesColor( Color.LTGRAY);
renderer.setLabelsColor( Color.LTGRAY);

XYMultipleSeriesDataset dataset = new XYMultipleSeriesDataset();
TimeSeries series = new TimeSeries("历史数据");

for (int k = 0; k < xlist.size(); k++) {
```




```

series.add(xlist.get(k), ylist.get(k));           //载入数据
}
dataset.addSeries(series);                       //通过 series 传递加载数据

GraphicalView mGrapView = ChartFactory.getTimeChartView(getBaseContext(),
dataset, renderer, "M/d-H:mm");
LinearLayout layout = (LinearLayout) findViewById(R.id.curveLayout);
LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
lp.weight = 1;
layout.addView(mGrapView, lp);                   //视图显示并加载

```

(8) 同理, 需要借助 try-catch 语句来处理查询失败情况。

```

try{
.....}
catch (Exception e) {
//TODO Auto-generated catch block
e.printStackTrace();
Toast.makeText(getApplicationContext(), "获取历史数据失败!",
Toast.LENGTH_SHORT).show();
}

```

3. 摄像头接口

(1) 实例化, 并初始化智云 ID 及密钥。

```

wCamera = new WSNCamera("12345678", "12345678"); //实例化, 并初始化智云 ID 及密钥

```

(2) 摄像头初始化, 并检测在线。

```

String myCameraIP = "ayari.easyn.hk";           //摄像头 IP
String user = "admin";                          //用户名
String pwd = "admin";                          //密码
String type = "H3-Series";                     //摄像头类型
wCamera.initCamera(myCameraIP, user, pwd, type);
mTVCamera.setText(myCameraIP + "正在检查是否在线...");
wCamera.checkOnline();

```

(3) 调用接口方法, 实现摄像头的控制。

```

public void onClick(View v) {
//TODO Auto-generated method stub
if (v == mBTNSnapshot) {
if(isOn==true)
{
wCamera.snapshot();
}
}
if (v == mBTNStartVideo) {
wCamera.openVideo();
mIVVideo.setVisibility(View.VISIBLE);
isOn=true;
}
if (v == mBTNStopVideo) {
wCamera.closeVideo();
mIVVideo.setImageBitmap(null);
}
}

```



```
mIVVideo.setVisibility(View.INVISIBLE);
isOn=false;
}
if (v == mBTNup) {
    if(isOn==true)
    {
        wCamera.control("UP");
    }
}
if (v == mBTNdown) {
    if(isOn==true)
    {
        wCamera.control("DOWN");
    }
}
if (v == mBTNleft) {
    if(isOn==true){
        wCamera.control("LEFT");}
}
if (v == mBTNright) {
    if(isOn==true){
        wCamera.control("RIGHT");}
}
if (v == mBTNHPatrol)
{
    if(isOn==true){
        wCamera.control("HPATROL");}
}
if (v == mBTNVPatrol) {
    if(isOn==true){
        wCamera.control("VPATROL");}
}
if (v == mBTN360Patrol) {
    if(isOn==true){
        wCamera.control("360PATROL");}
}
}
```

(4) 通过回调函数，返回 Bitmap，获取到拍摄图片。

```
public void onVideoCallBack(String camera, Bitmap bmp) {
    //TODO Auto-generated method stub
    if (camera.equals(myCameraIP)) {
        if(isOn==ture){
            mIVVideo.setImageBitmap(bmp);
        }
    }
}
```

(5) 释放摄像头资源。

```
public void onDestroy() {
    //释放摄像头资源
    wCamera.freeCamera();
}
```



```
super.onDestroy();
```

4. 应用数据接口

(1) 同样方法, 初始化 ID、key, 并建立连接, 连接服务器。

(2) 调用 wsnProperty 的 put(key,value)方法保存键值对。

```
String propertyKey = editKey.getText().toString();
String propertyValue = editValue.getText().toString();
if(propertyKey.equals("") || propertyValue.equals("")){
    Toast.makeText(PropertyActivity.this, "应用属性名或应用属性值不能为空",
        Toast.LENGTH_SHORT).show();
}else{
    try {
        wsnProperty.put(propertyKey, propertyValue);
        Toast.makeText(PropertyActivity.this, "成功保存应用属性值到服务器",
            Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(3) 调用 wsnProperty 的 get()方法读取键值对。

```
String propertyKey = editKey.getText().toString();
try {
    if(propertyKey.equals("")){
        String result = wsnProperty.get();
        Toast.makeText(PropertyActivity.this, "成功从服务器读取所有的应用属性值",
            Toast.LENGTH_SHORT).show();
        tvResult.setText(jsonFormatter(result));
    }else{
        String result = wsnProperty.get(propertyKey);
        Toast.makeText(PropertyActivity.this, "成功从服务器读取应用属性值",
            Toast.LENGTH_SHORT).show();
        tvResult.setText("属性名为: "+propertyKey+", 属性值为:
            "+jsonFormatter(result));
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

5. 自动控制接口

本任务中单独一个包作为示例, AutoControlActivity.java 是包中的主 Activity, 通过 button 跳转到 4 个不同的 Activity 中。下面对每个 Activity 进行详细阐述。

(1) TriggerActivity 是触发器的处理界面, 保存触发器基本信息 (name、MAC 地址、通道名、条件), 当传感器达到触发条件时, 进行执行器中的执行命令, 也可查询当前保存的触发器。

① 实例化, 并建立连接。

```
wsnAutoControl = new WSNAutoctrl(DemoActivity.myZCloudID,
    DemoActivity.myZCloudKey); //DemoActivity中定义的ID、key
wsnAutoControl.setServerAddr("zhiyun360.com:8001"); //用户自己设置
```




② 条件运算符选择。

```
radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        //TODO Auto-generated method stub
        if (checkedId == radioButton0.getId()) {
            operateSelected = radios[0];
        } else if (checkedId == radioButton1.getId()) {
            operateSelected = radios[1];
        } else if (checkedId == radioButton2.getId()) {
            operateSelected = radios[2];
        } else if (checkedId == radioButton3.getId()) {
            operateSelected = radios[3];
        } else if (checkedId == radioButton4.getId()) {
            operateSelected = radios[4];
        } else if (checkedId == radioButton5.getId()) {
            operateSelected = radios[5];
        } else if (checkedId == radioButton6.getId()) {
            operateSelected = radios[6];
        } else if (checkedId == radioButton7.getId()) {
            operateSelected = radios[7];
        } else if (checkedId == radioButton8.getId()) {
            operateSelected = radios[8];
        }
    }
});
```

③ 调用 `wsnAutoControl.createTrigger(name, "sensor", param)` 方法实现保存触发器信息。

```
JSONObject param = new JSONObject();
param.put("mac", mac);
param.put("ch", channel);
param.put("op", operateSelected);
param.put("value", operateValue);
param.put("once", true);
String result = wsnAutoControl.createTrigger(name, "sensor", param);
if (!result.equals("")) {
    JSONObject object = new JSONObject(result);
    Integer id = object.getInt("id");
    Toast.makeText(TriggerActivity.this, "添加触发器到服务器成功, 返回 ID 为" + id,
        Toast.LENGTH_SHORT).show();
} else {
    Toast.makeText(TriggerActivity.this, "添加触发器到服务器失败!",
        Toast.LENGTH_SHORT).show();
}
```

④ 调用 `wsnAutoControl.getTrigger()` 方法用来获取所有保存的触发器, 代码参考 `libwsnDrionDemo/src/TriggerActivity.java`。

⑤ 同理, 所有保存触发器和查询触发器的操作都会抛出异常, 因此要用 `try-catch` 语句进行处理。

⑥ 断开连接。



```
protected void onDestroy() {
    //TODO Auto-generated method stub
    super.onDestroy();
}
```

(2) **ActuatorActivity** 是执行器处理界面, 保存执行器基本信息, 用于相应触发器的条件处理事件, 执行命令。处理执行器和处理触发器的方法类似, 两者的主要区别在于方法的名称不同, 调用 **wsnAutoControl.createActuator(name,"sensor",param)** 方法来保存执行器信息, **wsnAutoControl.getActuator()** 方法来查询保存的执行器信息, 源码如下。

```
protected void onCreate(Bundle savedInstanceState) {
    //TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.actuator);
    setTitle("自动控制-执行器");
    editName = (EditText) findViewById(R.id.editName);
    editMac = (EditText) findViewById(R.id.editMac);
    editCommand = (EditText) findViewById(R.id.editCommand);
    btnSave = (Button) findViewById(R.id.btnSave);
    btnQuery = (Button) findViewById(R.id.btnQuery);
    tvResult = (TextView) findViewById(R.id.tvResult);

    wsnAutoControl = new WSNAutoctrl(DemoActivity.myZCloudID,
                                     DemoActivity.myZCloudKey);
    wsnAutoControl.setServerAddr("zhiyun360.com:8001");
    btnSave.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //TODO Auto-generated method stub
            String name = editName.getText().toString();
            String mac = editMac.getText().toString();
            String command = editCommand.getText().toString();
            if(name.equals("")||mac.equals("")||command.equals("")){
                Toast.makeText(ActuatorActivity.this, "执行器名或 MAC 地址或指令不  
能为空!", Toast.LENGTH_SHORT).show();
            }else{
                try {
                    System.out.println("in try");
                    JSONObject param = new JSONObject();
                    param.put("mac", mac);
                    param.put("data", command);
                    String result = wsnAutoControl.createActuator(name, "sensor",
param);

                    if(!result.equals("")){
                        JSONObject object = new JSONObject(result);
                        Integer id = object.getInt("id");
                        Toast.makeText(ActuatorActivity.this, "保存执行器到服务器成功,  
返回 ID 为"+id, Toast.LENGTH_SHORT).show();
                    }else{
                        Toast.makeText(ActuatorActivity.this, "保存执行器到服务器失败!",
Toast.LENGTH_SHORT).show();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    });
}
```



```
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

});

btnQuery.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        try {
            String result = wsnAutoControl.getActuator();
            tvResult.setText(jsonFormatter(result));
            Toast.makeText(ActuatorActivity.this, "查询所有的执行器成功!",
                Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
}

public String jsonFormatter(String uglyJSONString) {
    Gson gson = new GsonBuilder().disableHtmlEscaping().
        setPrettyPrinting().create();

    JsonParser jp = new JsonParser();
    JsonElement je = jp.parse(uglyJSONString);
    String prettyJsonString = gson.toJson(je);
    return prettyJsonString;
}

@Override
protected void onDestroy() {
    //TODO Auto-generated method stub
    super.onDestroy();
}
}
```

(3) `JobActivity` 是执行策略处理界面，用于匹配触发器和执行器，实现自动控制。调用 `wsnAutoControl.createJob(name,enable,param)` 方法用于创建执行策略，调用 `wsnAutoControl.getJob()` 方法用于浏览所有执行策略，源码如下。

```
protected void onCreate(Bundle savedInstanceState) {
    //TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.job);
    setTitle("自动控制-执行策略");
    editName = (EditText) findViewById(R.id.editName);
    editTriggerId = (EditText) findViewById(R.id.editTriggerID);
```




```

editActuatorId = (EditText) findViewById(R.id.editActuatorID);
radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
radioButton0 = (RadioButton) findViewById(R.id.radioGroupButton0);
radioButton1 = (RadioButton) findViewById(R.id.radioGroupButton1);
btnSave = (Button) findViewById(R.id.btnSave);
btnQuery = (Button) findViewById(R.id.btnQuery);
tvResult = (TextView) findViewById(R.id.tvResult);

wsnAutoControl = new WSNAutoctrl(DemoActivity.myZCloudID,
                                   DemoActivity.myZCloudKey);
wsnAutoControl.setServerAddr("zhiyun360.com:8001");

radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        //TODO Auto-generated method stub
        if (checkedId == radioButton0.getId()) {
            enable = true;
        } else if(checkedId == radioButton1.getId()){
            enable = false;
        }
    }
});

btnSave.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        String name = editName.getText().toString();
        String triggerId = editTriggerId.getText().toString();
        String actuatorId = editActuatorId.getText().toString();
        if(name.equals("")||triggerId.equals("")||actuatorId.equals("")){
            Toast.makeText(JobActivity.this, "执行策略名或触发器 ID或执行器 ID 不能为空！
            ", Toast.LENGTH_SHORT).show();
        }else{
            String[] tids = triggerId.split(",");
            String[] aids = actuatorId.split(",");
            try{
                JSONObject param = new JSONObject();
                JSONArray tidsArray = new JSONArray();
                JSONArray aidsArray = new JSONArray();
                for(int i=0;i<tids.length;i++){
                    tidsArray.put(tids[i]);
                }
                for(int j=0;j<aids.length;j++){
                    aidsArray.put(aids[j]);
                }
                param.put("tids", tidsArray);
                param.put("aids", aidsArray);
            }
        }
    }
});

```



```
String result = wsnAutoControl.createJob(name, enable, param);
if(!result.equals("")){
    JSONObject jsonObject = new JSONObject(result);
    Integer id = jsonObject.getInt("id");
    Toast.makeText(JobActivity.this, "保存执行策略到服务器成功, 返回ID为
        "+id, Toast.LENGTH_SHORT).show();
}else{
    Toast.makeText(JobActivity.this, "保存执行策略到服务器失败!",
        Toast.LENGTH_SHORT).show();
}
} catch (Exception e) {
    //TODO: handle exception
    Toast.makeText(JobActivity.this, "请输入正确的触发器ID和执行ID格式!",
        Toast.LENGTH_SHORT).show();
}
}
}
});

btnQuery.setOnClickListener(new View.OnClickListener() {

@Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        String result;
        try {
            result = wsnAutoControl.getJob();
            tvResult.setText(jsonFormatter(result));
        } catch (Exception e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
});

}

public String jsonFormatter(String uglyJSONString) {
    Gson gson = new GsonBuilder().disableHtmlEscaping().
        setPrettyPrinting().create();

    JsonParser jp = new JsonParser();
    JsonElement je = jp.parse(uglyJSONString);
    String prettyJsonString = gson.toJson(je);
    return prettyJsonString;
}

@Override
protected void onDestroy() {
    super.onDestroy();
}
}
```



(4) SchedudlerActivity 定义了用户查询执行记录的方法, 用户查询分为两种: 过滤查询和执行查询, 调用 `wsnAutoControl.getSchedudler(number, duration)` 方法用来过滤查询, 调用 `wsnAutoControl.getSchedudler()` 方法用来执行查询, 源码如下。

```
protected void onCreate(Bundle savedInstanceState) {
    //TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.schedudler);
    setTitle("自动控制-执行记录");
    wsnAutoControl = new WSNAutoctrl(DemoActivity.myZCloudID,
                                     DemoActivity.myZCloudKey);
    wsnAutoControl.setServerAddr("zhiyun360.com:8001");

    btnQuery = (Button) findViewById(R.id.btnQuery);
    btnFilter = (Button) findViewById(R.id.btnFilter);
    editNumber = (EditText) findViewById(R.id.editNumber);
    tvResult = (TextView) findViewById(R.id.tvResult);
    spinner = (Spinner) findViewById(R.id.spinnerDuration);
    //设置下拉列表的风格
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                                                         Android.R.layout.simple_spinner_item, durations);
    adapter.setDropDownViewResource(Android.R.layout.
                                     simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);

    spinner.setOnItemClickListener(new OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parent, View view,
                                   int position, long id) {
            //TODO Auto-generated method stub
            duration = durations[position];
        }

        @Override
        public void onNothingSelected(AdapterView<?> parent) {
            //TODO Auto-generated method stub
        }
    });

    btnFilter.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //TODO Auto-generated method stub
            String number = editNumber.getText().toString();
            if(editNumber.equals("")){
                Toast.makeText(SchedudlerActivity.this, "过滤查询的数值文本框
                不能为空!", Toast.LENGTH_SHORT).show();
            }else{
                String result;
                try {
                    result = wsnAutoControl.getSchedudler(number, duration);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    });
}
```




```

        tvResult.setText(jsonFormatter(result));
        Toast.makeText(SchedudlerActivity.this, "过滤查询执行记录成功!",
            Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

});

btnQuery.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        String result;
        try {
            result = wsnAutoControl.getSchedudler();
            tvResult.setText(jsonFormatter(result));
            Toast.makeText(SchedudlerActivity.this, "从服务器获取执行记录成功!",
                Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            //TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

});

}

public String jsonFormatter(String uglyJSONString) {
    Gson gson = new GsonBuilder().disableHtmlEscaping().
        setPrettyPrinting().create();

    JsonParser jp = new JsonParser();
    JsonElement je = jp.parse(uglyJSONString);
    String prettyJsonString = gson.toJson(je);
    return prettyJsonString;
}

@Override
protected void onDestroy() {
    //TODO Auto-generated method stub
    super.onDestroy();
}

}

```

2.4.5 开发步骤

按照任务 7 完成智云硬件环境搭建, 用 Android 集成开发环境打开 Android 开发例程: 在 eclipse 中导入开发例程 libwsnDroidDemo 文件, 目录为“04-开发例程\Chapter02 智云物联网开发基础\任务 06-智云 Android 简单应用libwsnDroidDemo”, 要实现 2.4.4 节中的所有内容都需要用到 libwsnDroid-20151103.jar 包中的 API, 因此, 需要将 libwsnDroid2.jar 包复制到工程目

录的 libs 文件夹下，如图 2.39 所示，运行调试无 bug。

正确填写智云 ID 密钥、服务器地址、摄像头信息。开发者智云 ID 密钥和服务器地址，摄像头信息有摄像头 IP、用户名、密码、摄像头类型，摄像头 IP 为摄像头连接网关后映射出的 IP，其他三个摄像头均已给出。将程序运行在 Android 终端，并组网成功。单击按钮，查看运行结果。

以实时连接接口和摄像头接口为例来显示运行结果。

(1) 主界面显示，分为多个模块，单击分别进入不同的模块，如图 2.40 所示。

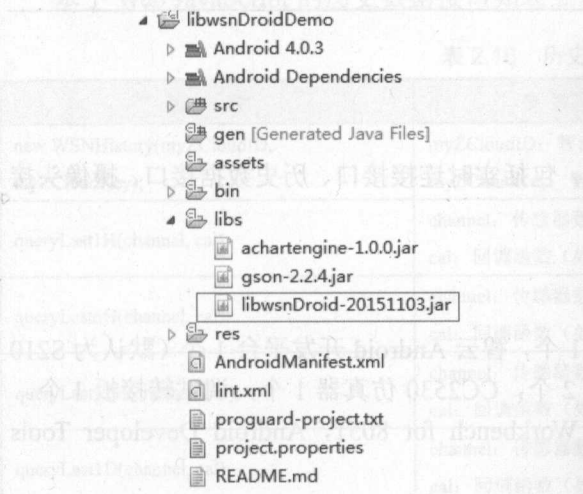


图 2.39 libs 文件夹下列表显示

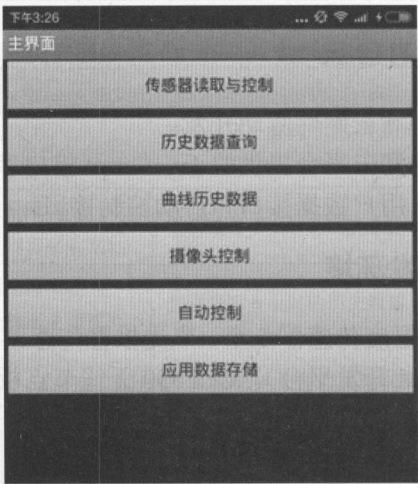


图 2.40 主界面

(2) 单击“传感器读取与控制”按钮，跳转到传感器读取与控制界面，此 Activity 调用的是实时连接接口中的方法，单击开灯、关灯，显示实时控制的指令输出，如图 2.41 所示。

(3) 返回到主界面，单击摄像头控制，进入摄像头控制模块，当显示出 ayari.easyn.hk（摄像头 IP）在线后，就可以通过按钮控制摄像头，如图 2.42 所示。

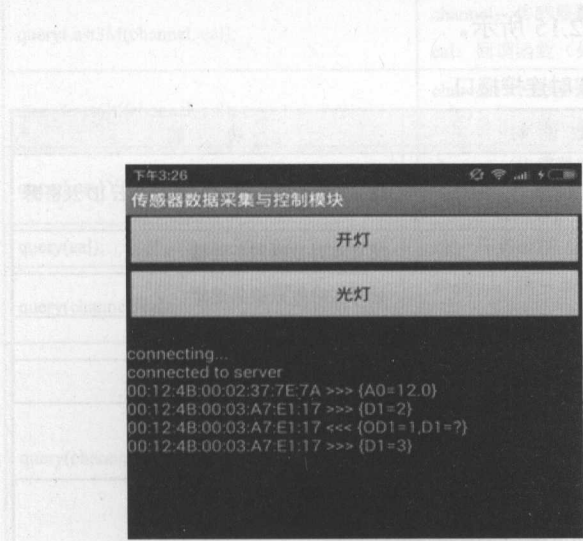


图 2.41 传感器读取与控制模块

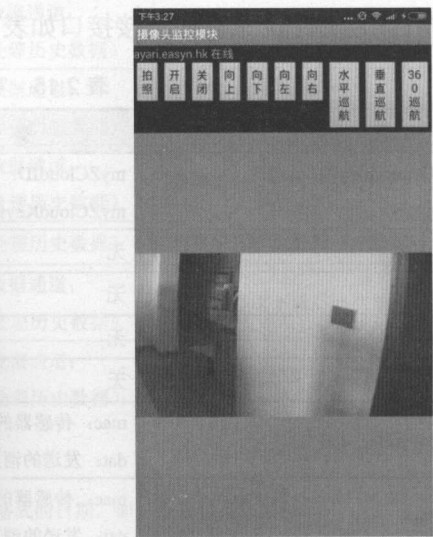


图 2.42 摄像头控制模块



2.4.6 总结与拓展

本任务详细介绍了 AndroidAPI 开发方法，开发者可以编写一个温湿度采集的应用，在主界面每隔 30 s 更新一次温湿度的值。

2.5 任务 9：WebAPI 开发

2.5.1 学习目标

- 掌握智云硬件驱动开发；
- 掌握智云平台提供的 JavaScript 接口库，包括实时连接接口、历史数据接口、摄像头接口、用户数据接口和自动控制接口。

2.5.2 开发环境

硬件：温度传感器 1 个，声光报警传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 2 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

2.5.3 原理学习

针对 Web 应用开发，智云平台提供 JavaScript 接口库，包括实时连接接口、历史数据接口、摄像头接口、用户数据接口和自动控制接口，开发者直接调用相应的接口完成简单 Web 应用的开发。

1. 实时连接接口

基于 Web JavaScript 的实时连接接口如表 2.15 所示。

表 2.15 实时连接接口

函 数	参 数 说 明	功 能
new WSNRTConnect(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	建立实时数据实例，并初始化智云 ID 及密钥
connect()	无	建立实时数据服务连接
disconnect()	无	断开实时数据服务连接
onConnect()	无	监听连接智云服务成功
onConnectLost()	无	监听连接智云服务失败
onMessageArrive(mac, dat)	mac: 传感器的 MAC 地址; dat: 发送的消息内容	监听收到的数据
sendMessage(mac, dat)	mac: 传感器的 MAC 地址; dat: 发送的消息内容	发送消息



续表

函 数	参 数 说 明	功 能
setServerAddr(sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥 (需要重新断开连接)

2. 历史数据接口

基于 Web JavaScript 的历史数据接口如表 2.16 所示。

表 2.16 历史数据接口

函 数	参 数 说 明	功 能
new WSNHistory(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化历史数据对象, 并初始化智云 ID 及密钥
queryLast1H(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 1 小时的历史数据
queryLast6H(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 6 小时的历史数据
queryLast12H(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 12 小时的历史数据
queryLast1D(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 1 天的历史数据
queryLast5D(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 5 天的历史数据
queryLast14D(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 14 天的历史数据
queryLast1M(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 1 月 (30 天) 的历史数据
queryLast3M(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 3 月 (90 天) 的历史数据
queryLast6M(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 6 月 (180 天) 的历史数据
queryLast1Y(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	查询最近 1 年 (365 天) 的历史数据
query(cal);	cal: 回调函数 (处理历史数据)	获取所有通道最后一次数据
query(channel, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据)	获取该通道下最后一次数据
query(channel, start, end, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据); start: 起始时间; end: 结束时间。 时间为 ISO 8601 格式的日期, 如 2010-05-20T11:00:00Z	通过起止时间查询指定时间段的历史数据



续表

函 数	参 数 说 明	功 能
query(channel, start, end, interval, cal);	channel: 传感器数据通道; cal: 回调函数 (处理历史数据); start: 起始时间; end: 结束时间; interval: 采样点的时间间隔, 详见后续说明。 时间为 ISO 8601 格式的日期, 如 2010-05-20T11:00:00Z	通过起止时间查询指定时间段指定时间间隔的历史数据
setServerAddr(sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(myZCloudID, myZCloudKey);	myZCloudID: 智云账号 myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥

3. 摄像头接口

基于 Web JavaScript 的摄像头接口如表 2.17 所示。

表 2.17 摄像头接口

函 数	参 数 说 明	功 能
new WSNCamera(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化摄像头对象, 并初始化智云 ID 及密钥
initCamera(myCameraIP, user, pwd, type);	myCameraIP: 摄像头外网域名/IP 地址; user: 摄像头用户名; pwd: 摄像头密码; type: 摄像头类型 (F-Series、F3-Series、H3-Series)。 以上参数从摄像头手册获取	设置摄像头域名、用户名、密码、类型等参数
openVideo();	无	打开摄像头
closeVideo();	无	关闭摄像头
control(cmd);	cmd: 云台控制命令, 参数如下: UP: 向上移动一次; DOWN: 向下移动一次; LEFT: 向左移动一次; RIGHT: 向右移动一次; HPATROL: 水平巡航转动; VPATROL: 垂直巡航转动; 360PATROL: 360° 巡航转动	发指令控制摄像头云台转动
checkOnline(cal);	cal: 回调函数 (处理检查结果)	检测摄像头是否在线
snapshot();	无	抓拍照片
setDiv(divID);	divID: 网页标签	设置展示摄像头视频图像的标签
freeCamera(myCameraIP);	myCameraIP: 摄像头外网域名/IP 地址	释放摄像头资源



续表

函 数	参 数 说 明	功 能
setServerAddr(sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥

4. 自动控制接口

基于 Web JavaScript 的自动控制接口如表 2.18 所示。

表 2.18 自动控制接口

函 数	参 数 说 明	功 能
new WSNAutoctrl(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化自动控制对象, 并初始化智云 ID 及密钥
createTrigger(name, type, param, cal);	name: 触发器名称; type: 触发器类型 (sensor、timer); param: 触发器内容, JSON 对象格式, 创建成功后返回该触发器 ID (JSON 格式); cal: 回调函数	创建触发器
createActuator(name, type, param, cal);	name: 执行器名称; type: 执行器类型 (sensor、ipcamera、phone、job); param: 执行器内容, JSON 对象格式创建成功后返回该执行器 ID (JSON 格式); cal: 回调函数	创建执行器
createJob(name, enable, param, cal);	name: 任务名称; enable: true (使能任务)、false (禁止任务); param: 任务内容, JSON 对象格式, 创建成功后返回该任务 ID (JSON 格式); cal: 回调函数	创建任务
deleteTrigger(id, cal);	id: 触发器 ID; cal: 回调函数	删除触发器
deleteActuator(id, cal);	id: 执行器 ID; cal: 回调函数	删除执行器
deleteJob(id, cal);	id: 任务 ID; cal: 回调函数	删除任务
setJob(id, enable, cal);	id: 任务 ID; enable: true (使能任务)、false (禁止任务); cal: 回调函数	设置任务使能开关
deleteSchedudler(id, cal);	id: 任务记录 ID; cal: 回调函数	删除任务记录
getTrigger(cal);	cal: 回调函数	查询当前智云 ID 下的所有触发器内容



续表

函 数	参 数 说 明	功 能
getTrigger(id, cal);	id: 触发器 ID; cal: 回调函数	查询该触发器 ID 内容
getTrigger(type, cal);	type: 触发器类型; cal: 回调函数	查询当前智云 ID 下的所有该类型的触发器内容
getActuator(cal);	cal: 回调函数	查询当前智云 ID 下的所有执行器内容
getActuator(id, cal);	id: 执行器 ID; cal: 回调函数	查询该执行器 ID 内容
getActuator(type, cal);	type: 执行器类型; cal: 回调函数	查询当前智云 ID 下的所有该类型的执行器内容
getJob(cal);	cal: 回调函数	查询当前智云 ID 下的所有任务内容
getJob(id, cal);	id: 任务 ID; cal: 回调函数	查询该任务 ID 内容
getSchedudler(cal);	cal: 回调函数	查询当前智云 ID 下的所有任务记录内容
getSchedudler(jid, duration, cal);	id: 任务记录 ID duration:duration=x<year month day hours minute> //默认返回 1 天的记录; cal: 回调函数	查询该任务记录 ID 某个时间段的内容
setServerAddr(sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥

5. 用户数据接口

基于 Web JavaScript 的用户数据接口如表 2.19 所示。

表 2.19 历史数据接口

函 数	参 数 说 明	功 能
new WSNProperty(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	初始化用户数据对象, 并初始化智云 ID 及密钥
put(key, value, cal);	key: 名称; value: 内容; cal: 回调函数	创建用户应用数据
get(cal);	cal: 回调函数	获取所有的键值对
get(key, cal);	key: 名称; cal: 回调函数	获取指定 key 的 value 值
setServerAddr(sa)	sa: 数据中心服务器地址及端口	设置/改变数据中心服务器地址及端口号
setIdKey(myZCloudID, myZCloudKey);	myZCloudID: 智云账号; myZCloudKey: 智云密钥	设置/改变智云 ID 及密钥



2.5.4 开发内容

1. 曲线的设计

在本任务中使用到的曲线图是采用了 highchart 公司提供的—个图表库，开发者在使用的时候，只需要在 HTML 中包含相关库文件，然后调用相关方法即可，图 2.43 所示是本例程中使用的一个曲线图样式。

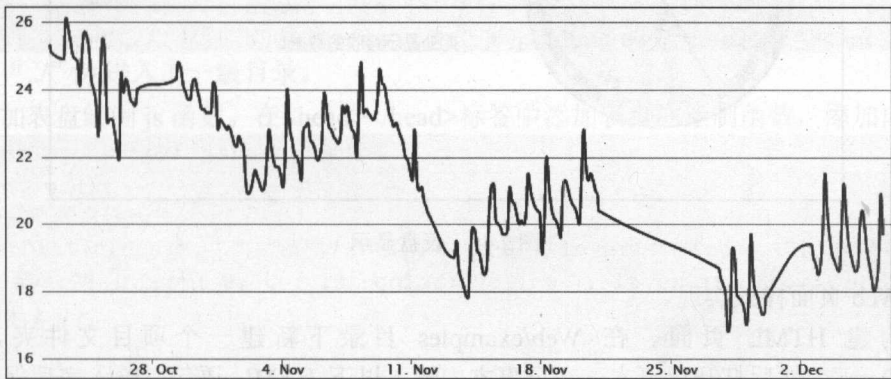


图 2.43 曲线图样式

曲线图说明：在图中横坐标表示为时间日期，纵坐标表示为显示的值。

2.5.5 节开发步骤曲线的实现中依次介绍在 HTML 页面中实现此曲线图的详细步骤。

2. 仪表的设计

在本任务中仪表的实现也是采用了 highchart 公司提供的—个图表库，开发者在使用的时候，只需要在 HTML 中包含相应的库文件，然后调用相应的方法即可，图 2.44 所示是本任务中使用的一个表盘样式。

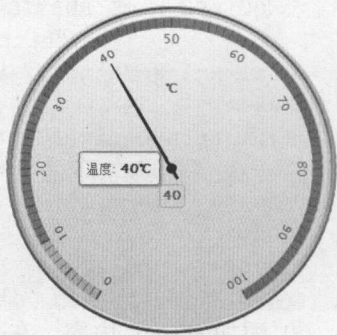


图 2.44 表盘样式

3. 实时数据的接收与发送

智云物联云平台提供了实时数据推送服务的 API，开发者根据这些 API 可以实现与底层传感器的信息交互，只有理解了这些 API 后，开发者才可以在底层自定义一些协议，然后根据 API 和协议就可以实现底层传感器的控制，数据采集等功能。

结合表盘和实时数据的发送与接收来实现任务，任务流程：Web 页面向底层硬件设备发送数据获取命令，底层设备成功收到命令后就上传相关数据，Web 页面接收到数据之后将数据进行解析之后就会在表盘中显示数据，同时在相应的标签中显示接收的原始数据。

下面实现基于 API 的 Web 页面。

(1) Web 页面的实现。在本任务中 realTimeData.html 页面的样式设计如图 2.45 所示，左侧的表盘显示温度值，右边通过文本框发送“{A0=?}”命令获取温度值，并将获取的温度值显示在表盘上，同时在下—方显示接收的数据，如图 2.45 所示。

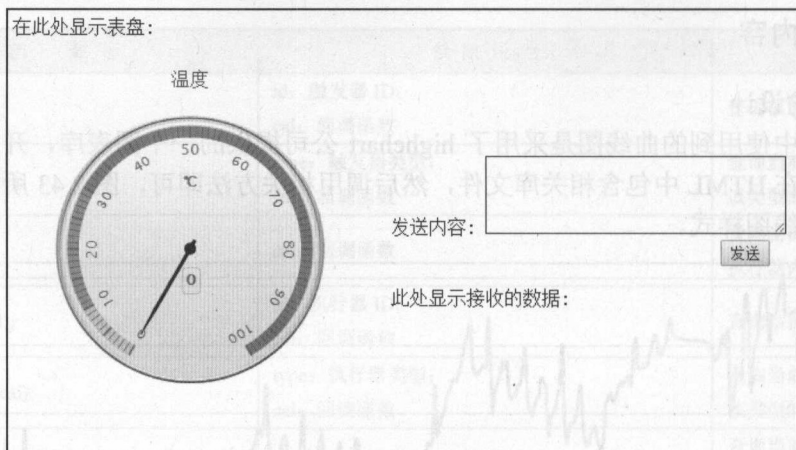


图 2.45 表盘显示

(2) Web 页面样式实现。

① 构建 HTML 页面。在 Web/examples 目录下新建一个项目文件夹，命名为“realTimeData”，然后打开记事本，在记事本中输入以下 HTML 语句，输入完后保存为 utf-8 格式，文件命名为 realTimeData.html，并将该文件保存到“Web/examples/realTimeData”文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>实时数据的发送与接收</title>
</head>
<body>
</body>
</html>
```

② 添加 HTML 内容。在<body></body>标签中添加 html 标签，内容为：

```
<div style = "margin-left:300px">
<div id = "state">连接中....</div>
<p>在此处显示表盘:</p>
<div id="dial" style = "width:300px;height:300px;float:left">
</div>
<div style="margin-left:320px;margin-top:100px;">
发送内容:
<textarea type="text" id="sendMessage" style=
    "width:250px;height:60px"></textarea><br>
<input type="button" value="发送" id="sendBt" style = "margin-left:280px">
<p>此处显示接收的数据:</p>
<div id = "showMessage"></div>
</div>
</div>
```




(3) Web 逻辑代码实现。

① 引入 js 脚本库。在 html 中添加 jQuery 语言库 jquery-1.11.0.min.js 和表盘实现的 highcharts.js 和 highcharts-more.js 库的引用, 然后再添加表盘控件绘制 API 的 drawcharts.js 文件以及实时数据推送服务的 WSNRTConnect.js 文件, 在<head></head>标签中添加如下代码。

```
<script src="../../js/jquery-1.11.0.min.js" type="text/javascript"></script>
<script src="../../js/highcharts.js" type="text/javascript"></script>
<script src="../../js/highcharts-more.js" type="text/javascript"></script>
<script src="../../js/drawcharts.js" type="text/javascript"></script>
<script src="../../js/WSNRTConnect.js" type="text/javascript"></script>
```

说明: “../” 为进入上一级目录。

② 添加表盘绘制 js 函数。在<head></head>标签中添加表盘的绘制函数, 添加内容如下。

```
<script type="text/javascript">
$(function(){
    //绘制表盘样式
    getDial("#dial", "", "温度", "°C", 0, 100, { layer1: { from: 10, to: 30, color:
green }, layer2: { from: 0, to: 10, color: yellow }, layer3: { from: 30, to: 100,
color: red } });
});
</script>
```

③ 添加实时数据连接的 js 代码。实现实时数据的发送与接收功能, 本任务例程以智云平台的一个测试案例的传感器进行测试, 实现流程如下: 创建数据服务对象→云服务初始化→发送命令数据→接收底层上传的数据→解析接收到的数据→数据显示。在上一步骤的表盘绘制代码后面添加如下 js 代码。

```
var myZCloudID = "123"; //序号
var myZCloudKey = "123"; //密钥
var mySensorMac = "00:12:4B:00:02:CB:A8:52"; //传感器的 MAC 地址
var rtc = new WSNRTConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
rtc.connect(); //数据推送服务连接

rtc.onConnect = function() { //连接成功回调函数
    $("#state").text("数据服务连接成功!");
};

rtc.onConnectLost = function() { //数据服务掉线回调函数
    $("#state").text("数据服务掉线!");
};

rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
    if((mac == mySensorMac) && (dat.indexOf(",") == -1)) { //接收数据过滤
        var recvMessage = mac + " 发来消息: " + dat;
        //给表盘赋值
        //将原始数据的数字部分分离出来
        dat = dat.substring(dat.indexOf("=")+1, dat.indexOf("}"));
        setDialData("#dial", parseFloat(dat)); //在表盘上显示数据
        $("#showMessage").text(recvMessage); //显示接收到的原始数据
    }
};

$("#sendBt").click(function() { //发送按钮单击事件
```



```
var message = $("#sendMessage").val();
rtc.sendMessage(mySensorMac, message); //向传感器发送数据
});
```

说明：在本任务中采用了智云物联云平台的一个测试案例中的温湿度传感器，其中 myZCloudID、myZCloudKey 为开发者注册云平台账户时用到的传感器序列号和密钥，mySensorMac 为传感器的 MAC 地址。开发者可以将 myZCloudID、myZCloudKey、channel 修改成自己的项目信息，然后实现自己项目案例中某个传感器的实时数据的发送与接收。

4. 历史数据的获取与展示

本任务是结合曲线完成的，在第一个任务中只是简单的实现了曲线图的绘制，并初始化了一些曲线数据值，本任务中的所有数据来自服务器，因此重点是实现如何从服务器获取数据，并将获取到的数据在曲线图上显示。

针对历史数据查询的需求，智云物联云平台提供了丰富的历史数据查询的 API，并将相应的方法封装到了 WSNHistory.js 文件中，开发者只需要调用 WSNHistory.js 中的若干方法即可实现历史数据的查询，下面根据这些 API 来实现历史数据查询的 Web 页面。

(1) 数据获取 API 使用示例。

① 查询最近 3 个月的历史数据。

```
var myZCloudID = "xxxx"; //开发者注册时使用的 ID
var myZCloudKey = "xxxx"; //密钥
var channel = 'MAC 地址_参数'; //数据通道，如 00:12:4B:00:02:CB:A8:52_A0
//新建一个对象，并初始化智云 ID、KEY
var WSNHistory = WSNHistory(myZCloudID, myZCloudKey);
//最近 3 个月历史数据查询
WSNHistory.queryLast3M(channel, function(data) { //data 参数为查询到的历史数据
    //数据处理
});
```

② 查询 2015 年 5 月 20 日~2015 年 6 月 20 日的历史数据。

```
var myZCloudID = "xxxx"; //开发者注册时使用的 ID
var myZCloudKey = "xxxx"; //密钥
var channel = 'MAC 地址_参数'; //数据通道，如 00:12:4B:00:02:CB:A8:52_A0
var WSNHistory = new WSNHistory(); //新建一个对象
WSNHistory.initZCloud(myZCloudID, myZCloudKey); //初始化
//查询 2015 年 5 月 20 日-2015 年 6 月 20 日
var startTime = "2014-12-22T15:52:28Z"
var endTime = "2015-05-22T15:52:28Z";
var interval = 1800;
//data 参数为查询到的历史数据
WSNHistory.query(startTime, endTime, interval, channel, function(data) {
    //数据处理
});
```

(2) Web 页面的实现。在本任务中 historyData.html 页面的样式设计如图 2.46 和图 2.47 所示，图 2.46 所示为历史数据曲线显示图，图 2.47 所示为所查询历史数据的原始数据显示栏。

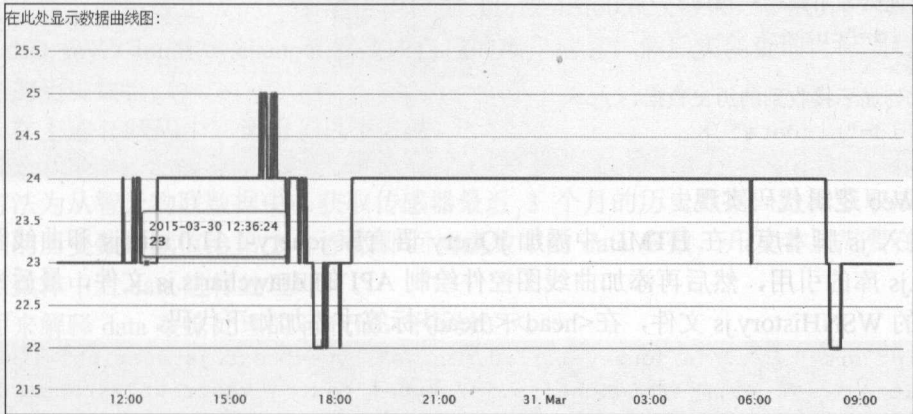


图 2.46 数据曲线显示图



图 2.47 历史数据显示

下面依次介绍在 HTML 页面中实现本例程的详细步骤。

(3) Web 页面样式实现。

① 构建 HTML 页面。在“Web/examples”目录下新建一个项目文件夹，命名为“historyData”，然后打开记事本，在记事本中输入以下 HTML 语句，输入完后保存为 utf-8 格式，文件命名为 historyData.html，并将该文件保存到“Web/examples/historyData”文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>历史数据的查询</title>
</head>
<body>
</body>
</html>
```

② 添加 HTML 标签。在<body></body>标签中添加 html 标签，内容如下所示。



```
<p>在此处显示数据曲线图:</p>
<div id="curve" >
</div>
<p>此处显示接收到的历史数据:</p>
<div id="hisData" >
</div>
```

(4) Web 逻辑代码实现。

① 引入 js 脚本库。在 HTML 中添加 jQuery 语言库 jquery-1.11.0.min.js 和曲线图实现的 highcharts.js 库的引用, 然后再添加曲线图控件绘制 API 的 drawcharts.js 文件, 最后添加历史数据查询的 WSNHistory.js 文件, 在<head></head>标签中添加如下代码。

```
<script src="../../../js/jquery-1.11.0.min.js" type="text/javascript"></script>
<script src="../../../js/highcharts.js" type="text/javascript"></script>
<script src="../../../js/drawcharts.js" type="text/javascript"></script>
<script src="../../../js/WSNHistory.js" type="text/javascript"></script>
```

说明: “../”为进入上一级目录。

② 编写 js 脚本调用曲线绘制的 API, 创建历史数据查询对象、云服务初始化、调用历史数据查询的 API, 然后将查询到的历史数据在曲线图上实现, 本任务例程以智云平台的一个测试案例的数据通道进行历史数据查询, 查询的数据时间为最近 1 天, 在<head></head>标签中添加如下 js 代码。

```
<script type="text/javascript">
$(function(){
    var myZCloudID ='123'; //开发者注册时使用的 ID
    var myZCloudKey ='123'; //密钥
    var channel='00:12:4B:00:02:CB:A8:52_A0'; //数据通道
    var myHisData = new WSNHistory(myZCloudID,myZCloudKey); //建立对象,并初始化

    //查询最近 3 个月的历史数据
    myHisData.queryLast3M(channel,function(dat){
        if(dat!="")
        {
            var str = JSON.stringify(dat); //将接收到的 json 数据对象转换成字符串
            $("#hisData").text(JSON.stringify(dat)); //显示接收到的原始数据

            //将接收到的 json 数据转换成二维数组形式在曲线图中显示
            var data = DataAnalysis(dat);
            showChart('#curve', 'spline', '', false,eval(data));
        }
        else
        {
            $("#curve").text("你查询的时间段没有数据!");
        }
    });
});
</script>
```

说明:

(a) 在本任务查询的历史数据源来自智云物联网云平台的一个测试案例, 其中 myZCloudID、myZCloudKey 为开发者注册云平台账户时用到的传感器序列号和密钥, channel 为传感器的



MAC 地址与上传参数组成的一个字符串,如 00:12:4B:00:02:CB:A8:52_A0。开发者可以将 myZCloudID, myZCloudKey, channel 修改成自己的项目信息,然后实现查询自己项目案例中某个传感器的历史数据。

(b) 在上述 js 代码中,调用了以下方法。

```
.queryLast3M(channel, function(data) { ... })
```

该方法为从智云物联数据中心获取传感器最近 3 个月的历史数据,数据获取成功后便会将获取到的历史数据赋值给第 2 个参数即回调函数中的 data 参数,开发者只需要在这个回调函数的函数体中对 data 进行处理即可。

下面来解释 data 数据处理的 js 代码,代码如下。

```
if(dat!="")
{
    var str = JSON.stringify(dat); //将接收到的 json 数据对象转换成字符串
    $("#hisData").text(JSON.stringify(dat)); //显示接收到的原始数据

    var data = DataAnalysis(dat); //将接收到的 json 数据转换成二维数组形式在曲线图中显示
    showChart('#curve', 'spline', '', false, eval(data));
}
else
{
    $("#curve").text("你查询的时间段没有数据!");
}
```

根据上述代码不难理解,当 data 数据不为空时,将 data 数据作为曲线图的绘制数据源,由于曲线图库需要的数据格式是二维数组格式,而从服务器获取到的数据是 json 格式的数据形式,所以需要将接收到的 json 格式数据转换成二维数组的格式。

在该代码中分别调用了 DataAnalysis() (该函数自定义,在 drawcharts.js 中)、eval() 函数(系统提供的函数)将 data 的数据格式转换成真正的二维数组。数据格式转换结束之后,调用曲线绘制函数在指定的 id=curve 标签中显示曲线图,同时在 id=hisData 标签中显示获取到的源历史数据内容;当 data 为空时,即显示“你查询的时间段没有数据!”。

③ 自定义获取历史数据。本任务中是以获取最近 1 天的历史数据为例,获取 1 天、5 天、2 周、1 个月等时间的历史数据方法类似,只需要将函数名更改一下即可,同时历史数据查询的 API 也支持自定义时间段的数据查询,下面介绍 query() 函数的使用方法,开发者只需要在 js 代码区写如下内容即可。

```
<script type="text/javascript">
$(function(){
    var myZCloudID = '123'; //开发者注册时使用的 ID
    var myZCloudKey = '123'; //密钥
    var channel = '00:12:4B:00:02:CB:A8:52_A0'; //数据通道
    var myHisData = new WSNHistory(myZCloudID, myZCloudKey); //建立对象,并初始化
    //任意时间段、时间间隔的历史数据查询
    var startTime = "2014-12-22T15:52:28Z";
    var endTime = "2015-05-22T15:52:28Z";
    var interval = 1800;
    myHisData.query(channel, startTime, endTime, interval, function(dat){
        if(dat!="")
        {
```



```

var str = JSON.stringify(dat); //将接收到的 json 数据对象转换成字符串
$("#hisData").text(JSON.stringify(dat)); //显示接收到的原始数据

//将接收到的 json 数据转换成二维数组形式在曲线图中显示
var data = DataAnalysis(dat);
showChart('#curve', 'spline', '', false, eval(data));

}
else
{
    $("#curve").text("你查询的时间段没有数据!");
}
});
});
</script>

```

5. 摄像头的显示与控制

智云物联云平台提供了 IP 摄像头的若干 API，开发者只要掌握了这些 API 的使用便可轻松掌握 Web 端视频监控的开发实现。在视频监控的实现中需要用到 camera-1.1.js 库文件、WSNCamera.js 文件，开发者用到的一些 API 都封装在 WSNCamera.js 中，而 WSNCamera.js 中的 API 的功能是依赖于 camera-1.0.js 库文件的，因此开发者在进行 Web 端视频监控开发时，需要引用这两个 js 文件，下面根据这些 API 来实现 Web 端视频监控的编写，检测摄像头是否在线 API 使用说明：使用过程中按照如下方法调用并判断摄像头是否在线。

```

WSNCamera.checkOnline(function(state){
    //状态处理
    //state=1,摄像头在线
    //state=0,摄像头离线
});

```

摄像头初始化工程中 myCameraIP 参数说明：该参数支持“Camera:[IP:端口号]”或者“Camera:[域名]”两种赋值方式。如果摄像头作为端口映射，可以实现外网访问，则推荐开发者将该参数赋值为“Camera:[域名]”的形式；若摄像头只能在局域网访问，则该参数应赋值为“Camera:[IP:端口号]”的形式。

(1) Web 页面的实现。在本任务中 ipCamera.html 页面的样式设计如图 2.48 所示，左边按钮区域为摄像头的控制按钮，右边为视频监控的显示区域。

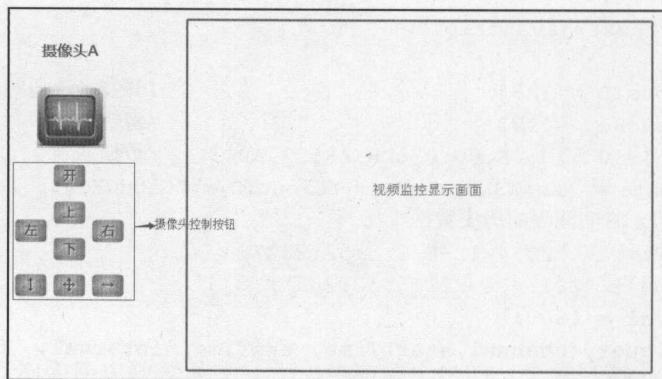


图 2.48 ipCamera.html 页面设计



下面详细介绍视频监控 Web 页面的开发过程。

(2) Web 页面样式实现。

① 构建 HTML 页面。在 Web/examples 目录下新建一个项目文件夹，命名为 ipCamera，然后打开记事本，在记事本中输入以下 HTML 语句，输入完后保存为 utf-8 格式，文件命名为 ipCamera.html，并将该文件保存到 Web/examples/ipCamera 文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>视频监控系统的实现</title>
</head>
<body>
</body>
</html>
```

② 添加 HTML 标签。在<body></body>标签中添加 HTML 标签，内容为：

```
<div id="header">
<div id="header_main">
<div id="logo" style="text-align:center;"></div>
<div id="banner">
<div id="logo_title">视频监控系统</div>
<div id="logo_intro">实现对家庭的视频监控以及远程控制。</div>
</div>
</div>
</div>
<div id="block"></div>
<div id="main">
<div class="clear"></div>
<div class="custom_box button-rounded glow_in">
<div class="c_box">
<div class="c_sensor_title">
<div class="c_sensor_name c_monitor_name">摄像头 A</div>
<div class="c_sensor_img"></div>

<div id="switch" class="switch_camera switch_camera_button"></div>
<div class="controller">
<div id="ct_up" class="ct_up switch_camera_button"></div>
<div id="ct_left" class="ct_left switch_camera_button"></div>
<div id="ct_right" class="ct_right switch_camera_button"></div>
<div id="ct_down" class="ct_down switch_camera_button"></div>
<div id="ct_h" class="ct_h switch_camera_button"></div>
<div id="ct_c" class="ct_c switch_camera_button"></div>
<div id="ct_v" class="ct_v switch_camera_button"></div>
</div>
</div>
<div class="c_sensor_box">
<div class="monitor_video button-rounded">
<img id="img1" border="0" src="">
</div>
```



```
</div>
</div>
<div class="clear"></div>
</div>
</div>
<div class="clear"></div>
```

③ 创建视频监控 Web 页面图片素材。开发者可以自定义，首先在 ipCamera 文件夹下创建 images 文件夹，然后在这个文件夹中存放参考页面展示的按钮图片等素材。

④ 编写 css 样式表。在第 3 步骤中添加了视频监控显示的 HTML 标签，但没有导入 css 样式表，为了让视频监控的 Web 页面显示与摄像头参考页面相同，需要编写 css 样式表，根据第③步骤编写的 HTML 标签以及参考页面的设计图来编写 custom.css 和 style.css 文件，两个 css 文件的内容如下所示。

custom.css 文件如下。

```
@charset "utf-8";
/*CSS Document */
.custom_box
{
    width:1000px;
    border:1px solid #ccc;
}
.custom_box .dot_line
{
    width:960px;
    height:10px;
    margin:auto;
}
.c_box
{
    width:960px;
    height:auto;
    margin:auto;
}
.c_sensor_title
{
    width:200px;
    height:auto;
    float:left;
}
.c_sensor_name
{
    width:200px;
    height:50px;
    line-height:50px;
    text-align:center;
    float:left;
    margin-top:20px;
    font-size:20px;
    font-weight:600;
```



```
        margin-left:10px;
    }
    .c_sensor_img
    {
        float:left;
        margin-left:48px;
    }
    .c_sensor_box
    {
        width:760px;
        height:auto;
        float:left;
    }
    .switch
    {
        margin-left:80px;
        background-image:url(../images/button_on.png);
    }
    .switch_camera
    {
        margin-left:88px;
        background-image:url(../images/button_on.png);
    }
    .switch_camera_button
    {
        width:44px;
        height:31px;
        float:left;
        cursor:pointer;
    }
    .monitor_video
    {
        width:640px;
        height:480px;
        margin:50px 60px;
        border:3px solid #666;
    }
    .c_monitor_name
    {
        margin-top:70px;
    }
    .controller
    {
        width:200px;
        height:auto;
        margin-top:20px;
        float:left;
        position:relative;
        margin-left:10px;
    }
```




```
.controller div
{
    position:absolute;
}
.ct_up
{
    background-image:url(../images/button_up.png);
    top:0;
    left:78px;
}
.ct_down
{
    background-image:url(../images/button_down.png);
    top:50px;
    left:78px;
}
.ct_left
{
    background-image:url(../images/button_left.png);
    top:25px;
    left:25px;
}
.ct_right
{
    background-image:url(../images/button_right.png);
    top:25px;
    right:25px;
}
.ct_v
{
    background-image:url(../images/button_vertical.png);
    top:100px;
    left:25px;
}
.ct_h
{
    background-image:url(../images/button_horizon.png);
    top:100px;
    right:25px;
}
.ct_c
{
    background-image:url(../images/button_comprehensive.png);
    top:100px;
    left:78px;
}
```

style.css 文件。

```
@charset "utf-8";
/*CSS Document */
```

```
/*=====主体设计=====*/
```



```
html
{
    margin:0;
    padding:0;
    overflow-y:scroll;
    overflow-x:hidden;
    background-color:#000;
}
body
{
    margin:0;
    padding:0;
    font-family:"微软雅黑", "黑体", "宋体";
    font-size:16px;
    color:#5D5A59;
    background-color:#fff;
}
img
{
    border:none;
}
.clear
{
    clear:both;
    height:30px;
}
.glow_in
{
    box-shadow: 0 0 10px #ccc inset;
    -webkit-box-shadow: 0 0 10px #ccc inset;
    -moz-box-shadow: 0 0 10px #ccc inset;
    background-color:#fefefe;
}
/*=====头部设计=====*/
#header
{
    height:145px;
}
#header_main
{
    width:1000px;
    height:145px;
    margin:auto;
    position:relative;
}
#logo
{
    background-repeat:no-repeat;
```



```
background-position:right;
width:200px;
height:80px;
position:relative;
float:left;
left:30px;
top:20px;
}
#banner
{
width:340px;
height:80px;
position:relative;
left:30px;
top:40px;
float:left;
display:block;
}
#logo_title
{
font-size:20px;
font-weight:600;
}

#logo_intro
{
font-size:12px;
margin-top:5px;
}
#block
{
height:10px;
background-color:#000;
}
/*=====中部设计=====*/
#main
{
width:1000px;
margin:auto;
min-height:300px;
}
```

⑤ 导入样式表。在 WSNCamera 文件夹下创建 css 文件夹，并将 custom.css 和 style.css 文件复制到 css 文件夹中，并在 ipCamera.html 中添加对其的引用。在<head></head>标签中添加如下内容。

```
<link href="css/style.css" rel="stylesheet" type="text/css" />
<link href="css/custom.css" rel="stylesheet" type="text/css" />
```




⑥ 上述步骤完成后，便实现了视频监控的 Web 页面编写，显示效果如图 2.49 所示。

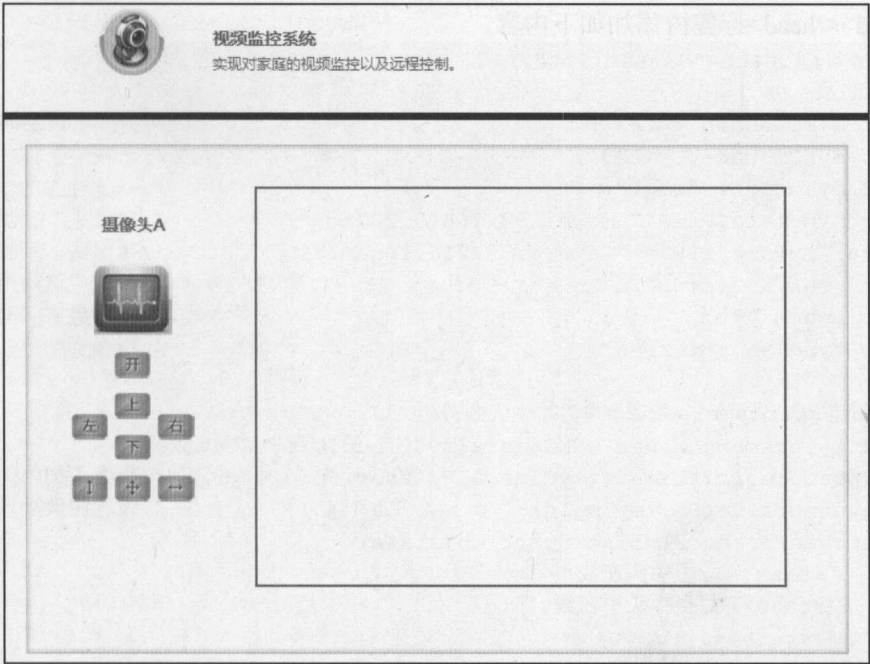


图 2.49 视频监控 Web 页面显示

(3) Web 逻辑代码实现。

① 引入 js 脚本库。在 html 中添加 jQuery 语言库 jquery-1.11.0.min.js，然后添加视频监控 API 的 WSNCamera.js 文件及 camera-1.1.js 文件，添加方法为在<head></head>标签中添加如下代码。

```
<script src="../../../js/jquery-1.11.0.min.js"></script>
<script src="../../../js/camera-1.1.js" type="text/javascript"></script>
<script src="../../../js/WSNCamera.js" type="text/javascript"></script>
```

说明：“../”为进入上一级目录。

② 在前面的一些步骤只是实现了页面的 HTML 代码编写，接下来重要的就是实现逻辑代码编写，以实现摄像头的显示，控制等功能。

编写摄像头 js 代码的流程如下：创建 WSNCamera 对象→云服务初始化→摄像头初始化→指定视频图像显示的位置→绑定摄像头的控制函数到控制按钮。摄像头的控制函数与控制按钮之间的绑定关系如表 2.20 所示。

表 2.20 按钮函数接口

按 钮	调用方法	按 钮	调用方法
开	openVideo();	关	closeVideo();
上	control("UP");	下	control("DOWN");
左	control("LEFT");	右	control("RIGHT");
↔	control("HPATROL");	↑	control("VPATROL");
⊕	control("360PATROL");		



根据上述流程及按钮与摄像头的控制函数的绑定关系来进行js 代码的编写, 编写过程如下。
在<head></head>标签内添加如下内容。

```
<script language="Javascript">
$(function () {
    var myZCloudID = '123'; //开发者注册时使用的 ID
    var myZCloudKey = '123'; //密钥
    var myImgId = "img1"; //img 标签的 ID
    var myCameraIP = "Camera:192.168.0.207:83"; //摄像头 IP 地址
    //var myCameraIP = "Camera:069210.ipcam.hk "; //摄像头 IP 地址
    var user = "admin"; //摄像头访问用户名
    var pwd = ""; //摄像头访问密码
    var type = "F-Series"; //摄像头型号

    //创建 myipcamera 对象, 并初始化云服务
    var myipcamera = new WSNCamera(myZCloudID, myZCloudKey);
    myipcamera.initCamera(myCameraIP, user, pwd, type); //摄像头初始化
    myipcamera.setDiv(myImgId); //设置图像显示的位置
    myipcamera.checkOnline(function(state){
        //state =1, 摄像头在线
        //state =0, 摄像头不在线
        if(state==1)
        {
            myipcamera.setResolution("640_480"); //将摄像头的分辨率设置成 640*480
        }
    });
    //初始化监视器开关默认关
    $("#switch").click(function () {
        if (!this.flag) {
            $(this).css("background-image", "url(images/button_off.png)");
            myipcamera.openVideo(); //打开摄像头并显示
            $("#imgSnapshot").click(function () {
                myipcamera.snapshot();
            });
        }
        else {
            $(this).css("background-image", "url(images/button_on.png)");
            myipcamera.closeVideo(); //关闭视频监控
        }
        this.flag = !this.flag;
    });
    //监视器控制器
    $("#ct_up").mousedown(function () { //上
        myipcamera.control("UP"); //向摄像头发送向上移动命令
        $(this).css("background-image", "url(images/button_up_disable.png)");
    }).mouseup(function () {
        $(this).css("background-image", "url(images/button_up.png)");
    }).mouseleave(function () {
        $(this).css("background-image", "url(images/button_up.png)");
    });
});
```



```
$("#ct_down").mousedown(function () { //下
    myipcamera.control("DOWN"); //向摄像头发送向下移动命令
    $(this).css("background-image", "url(images/button_down_disable.png)");
}).mouseup(function () {
    $(this).css("background-image", "url(images/button_down.png)");
}).mouseleave(function () {
    $(this).css("background-image", "url(images/button_down.png)");
});

$("#ct_left").mousedown(function () { //左
    myipcamera.control("LEFT"); //向摄像头发送向左移动命令
    $(this).css("background-image", "url(images/button_left_disable.png)");
}).mouseup(function () {
    $(this).css("background-image", "url(images/button_left.png)");
}).mouseleave(function () {
    $(this).css("background-image", "url(images/button_left.png)");
});

$("#ct_right").mousedown(function () { //右
    myipcamera.control("RIGHT"); //向摄像头发送向右移动命令
    $(this).css("background-image", "url(images/button_right_disable.png)");
}).mouseup(function () {
    $(this).css("background-image", "url(images/button_right.png)");
}).mouseleave(function () {
    $(this).css("background-image", "url(images/button_right.png)");
});

$("#ct_h").mousedown(function () { //水平巡航
    myipcamera.control("HPATROL"); //向摄像头发送水平巡航命令
    $(this).css("background-image", "url(images/button_horizon_disable.png)");
}).mouseup(function () {
    $(this).css("background-image", "url(images/button_horizon.png)");
}).mouseleave(function () {
    $(this).css("background-image", "url(images/button_horizon.png)");
});

$("#ct_v").mousedown(function () { //垂直巡航
    myipcamera.control("VPATROL"); //向摄像头发送垂直巡航命令
    $(this).css("background-image", "url(images/button_vertical_disable.png)");
}).mouseup(function () {
    $(this).css("background-image", "url(images/button_vertical.png)");
}).mouseleave(function () {
    $(this).css("background-image", "url(images/button_vertical.png)");
});

$("#ct_c").mousedown(function () { //360° 巡航
    myipcamera.control("360PATROL"); //向摄像头发送 360° 巡航命令
    $(this).css("background-image",
        "url(images/button_comprehensive_disable.png)");
```




```
}).mouseup(function () {
    $(this).css("background-image", "url(images/button_comprehensive.png)");
}).mouseleave(function () {
    $(this).css("background-image", "url(images/button_comprehensive.png)");
});
});
</script>
```

在初始化创建 WSNCamera 对象时,摄像头采用了智云物联云平台测试案例中的一个摄像头,并且在测试时采用了局域网内测试,开发者若要实现摄像头的视频监控,另外加上 IP 摄像头,并按照摄像头部署方法进行摄像头部署(参考 <http://www.zhiyun360.com/Home/Post/18>),获取摄像头的 IP 地址、端口号、访问账号、密码、摄像头类型之后再参考如下形式修改即可:

```
//摄像头 IP 地址+端口号,修改成自己部署的摄像头 IP 地址、端口号
var myCameraIP = "Camera:192.168.0.207:83";
var user = "admin"; //修改成部署摄像头的访问用户名
var pwd = ""; //修改成部署摄像头访问密码
var type = "H3-Series"; //摄像头型号
```

6. 用户应用数据开发

(1) Web 页面的实现。在本任务中 property.html 页面的样式设计如下图所示,第 1 行为用户应用数据模块应用 ID、应用 KEY、服务器地址配置栏,第 2 栏用户应用数据查询栏,第 3 栏为用户应用数据创建栏,第 4 栏为用户应用数据显示栏,如图 2.50 所示。

应用数据查询与创建

应用ID

1155223953

密钥

Xrk6UicNrbo3Kix1tYDDaUq9HAMH

服务器地址

1.zhiyun360.com:8080

应用名称

查询

(应用名称为空时查询所有应用信息,输入名称后查询指定应用信息)

应用名称

应用值

创建任务

图 2.50 应用数据查询与创建

(2) Web 页面样式实现。

① 构建 HTML 页面。在“Web/examples”目录下新建一个项目文件夹,命名为 property,然后打开记事本,在记事本中输入以下 html 语句,输入完后保存为 utf-8 格式,文件命名为 property.html,并将该文件保存到“Web/examples/property”文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>应用数据查询与创建</title>
</head>
<body>
</body>
</html>
```

② 添加 html 标签。在<body></body>标签中添加 html 标签,内容如下所示。



```

<h2>应用数据查询与创建</h2>
<!--配置服务器信息-->
<div class="server block">
<label for="aid">应用 ID</label>
<input type="text" id="aid" value="123"/>
<label for="xkey">密钥</label>
<input type="text" id="xkey" class="form-control" value="123" />
<label for="saddr">服务器地址</label>
<input type="url" id="saddr" class="form-control" value="zhiyun360.com:8080"/>
</div>
<hr/>
<!--查询应用数据查询-->
<div class="query block">
<label for="name">应用名称</label>
<input type="text" id="name" class="form-control" />
<input type="button" id="property_query" value="查询" /> (应用名称为空时查询所有
应用信息, 输入名称后查询指定应用信息)
</div>
<hr/>
<!--创建应用数据-->
<div class="create block">
<label for="property_name">应用名称</label>
<input type="text" id="property_name" class="form-control"/>
<label for="property_value">应用值</label>
<input type="text" id="property_value" class="form-control"/>
<input type="button" id="property_create" class="form_button" value="创建任
务" />
</div>
<hr/>
<div id="data_show" class="data block">
<!--显示操作后数据-->
</div>

```

③ 编写 css 样式。为了使得 property.html 排版更加美观, 需要编写 css 样式, 根据第 2 步骤编写的 HTML 标签及 property.html 参考页面的设计图, 在<head></head>中添加如下 css 样式代码。

```

<style>
h2{
    text-align:center;
}
hr{
    width:1024px;
    margin:0px auto;
    border: 1px dashed #666;
}
label{
    width: 80px;
}
.form-control{
    width: 80px;
}

```



```

margin-top:10px;
}
.form_button{
margin-top:10px;
}
.server .form-control{
width: 200px;
}
.block{
width:1024px;
margin:10px auto;
}
</style>

```

(3) Web 逻辑代码实现。

① 引入 js 脚本库。添加“jquery-1.11.0.min.js”和“WSNProperty.js”脚本库文件，添加方法为在<head></head>标签中添加如下代码。

```

<script src="../../WSN/jquery-1.11.0.min.js"></script>
<script src="../../WSN/WSNProperty.js"></script>

```

说明：“../”为进入上一级目录。

② 编写 js 逻辑代码。

```

<script>
$(function(){
//获取资产信息
$("#property_query").click(function(){
//初始化 api
var myProperty = new WSNProperty($("#aid").val(), $("#xkey").val());
myProperty.setServerAddr($("#saddr").val());
var name = $("#name").val();
if(name.length>0){
myProperty.get(name, function(dat){
$("#data_show").text(dat);
});
}else{
myProperty.get(function(dat){
$("#data_show").text(dat);
});
}
})
//创建资产信息
$("#property_create").click(function(){
//初始化 api
var myProperty = new WSNProperty($("#aid").val(), $("#xkey").val());
myProperty.setServerAddr($("#saddr").val());
var name = $("#property_name").val();
var val = $("#property_val").val();
myProperty.put(name,val, function(dat){
var data = JSON.stringify(dat);
$("#data_show").text(data);
});
});

```

//json 对象变为字符串



```
    })
  })
</script>
```

7. 自动控制模块开发

智云物联云平台用户项目中的自动控制模块具有触发器、执行器、执行任务、执行历史记录等 API，这些 API 全部封装在 WSNAutoctrl.js 文件中，使用时导入该包即可。

(1) 创建触发器详细说明。

方法：createTrigger(name, type, param, cb);

- name 参数：触发器名。
- type 参数：触发器类型，可取值[sensor|timer]。
- cb 参数：数据处理回调函数。
- param 参数：触发器实体依据 type 的不同，其实体参数取值也会有所不同，如下所示。当 type="sensor"时，其 param 参数取值格式如下。

```
"param":{
  "uid":"<应用 ID>",          //可选参数，默认使用 url 中应用 ID，否则通过 uid 指定应用 ID
  "mac":"<节点 MAC 地址>",
  "ch":"<通道名>",
  "op":"<比较运算符>",        //op 取值：>,>=,<,<=,==,!=,CHANGE,&,!&
  "value":"<通道值>",
  "once":true                  //once:true 第一次触发，false 每次触发
}
```

当 type="timer"时，其 param 参数取值格式如下。

```
"param":{
  "year":"*",                  //指定年
  "month":"*",                 //指定月
  "day":"*",                   //指定日
  "week":"*",                  //指定星期
  "hour":"7",                  //指定小时
  "minute":"1",                //指定分钟
  "second":"0"                 //指定秒
}
```

① 创建执行器详细说明。

方法：createActuator(name, type, param, cb);

- name 参数：执行器名。
- type 参数：执行器类型，可取值为[sensor|ipcamera|phone|job]。
- cb 参数：数据处理回调函数。
- param 参数：执行器实体依据 type 的不同，其实体参数取值也会有所不同，如下所示。当 type="sensor"时，其 param 参数取值格式如下。

```
"param":{
  "uid":"<应用 ID>",          //可选参数，默认使用 url 中应用 ID，否则通过 uid 指定应用 ID
  "mac":"<节点 MAC 地址>",
  "data":"<节点指令>"
}
```

当 type="ipcamera"时，其 param 参数取值格式如下。

```
"param":{
```



```
"mac": "Camera:069219.ipcam.hk",
"user": "<用户>",
"pwd": "<密码>",
"type": "<摄像头类型>",
"data": "{Action=TakenPicture}",
}
```

当 type="phone"时，其 param 参数取值格式如下。

```
"param": {
  "uid": "<应用 ID>", //可选参数，默认使用 url 中应用 ID，否则通过 uid 指定应用 ID
  "mac": "Phone:xxxxx",
  "data": "{Number=22222,Action=[SendSMS],[Content=xxxxxxx]}"
}
```

当 type="job"时，其 param 参数取值格式如下。

```
"param": {
  "uid": "<应用 id>", //可选参数，默认使用 url 中应用 ID，否则通过 uid 指定应用 ID
  "jid": "<job id>",
  "enable": true|false
}
```

② 创建执行任务详细说明。

方法：WSNAutoctrl.createJob(name, enable, param, cb);

- name 参数：执行器名。
- enable 参数：是否执行标志符。
- cb 参数：数据处理回调函数。
- param 参数：格式如下所示。

```
"param": {
  "tids": [<触发器 id>, ...],
  "aids": [<执行器 id>, ...],
}
```

(2) 触发器。在本任务中 auto_sensor.html 页面的样式设计如下图所示，第 1 行为自动控制触发器应用 ID、应用 KEY、服务器地址配置，第 2 栏触发器查询栏，第 3 栏为触发器（传感器类型）创建栏，第 4 栏为触发器（定时器类型）创建栏，第 5 栏为查询触发器显示栏，如图 2.51 所示。

自动控制触发器演示

应用ID

1155223953

密钥

Xrk6UicNrbo3Kix1tYDDaUq9HAMF

服务器地址

t.zhiyun360.com:8001

触发器ID

查询

(触发器ID为空时查询所有触发器信息，输入ID后查询指定触发器信息)

触发器名

触发器类型

sensor

mac地址

通道

A0

计算

大于

值

触发规则

第一次触

创建传感器类型触发器

触发器名

触发器类型

timer

年

月

周

周几

日

时

分

秒

创建定时器类型触发器

(创建定时器类型触发器是，值为“*/?”格式，表示每隔多久触发。)

图 2.51 自动控制触发器显示



(3) Web 页面样式实现。

① 构建 HTML 页面。在 Web/examples 目录下新建一个项目文件夹，命名为 autoctrl，然后打开记事本，在记事本中输入以下 HTML 语句，输入完后保存为 utf-8 格式，文件命名为 auto_sensor.html，并将该文件保存到 Web/examples/autoctrl 文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>自动控制触发器</title>
</head>
<body>
</body>
</html>
```

② 添加 HTML 标签。在<body></body>标签中添加 HTML 标签，内容如下所示。

```
<h2>自动控制触发器演示</h2>
<!--配置服务器信息-->
<div class="server block">
<label for="aid">应用 ID</label>
<input type="text" id="aid" value="123"/>
<label for="xkey">密钥</label>
<input type="text" id="xkey" class="form-control" value="123" />
<label for="saddr">服务器地址</label>
<input type="url" id="saddr" class="form-control" value="zhiyun360.com:8001"/>
</div>
<hr/>
<!--查询触发器-->
<div class="query block">
<label for="sensor_id">触发器 ID</label>
<input type="text" id="sensor_id" class="form-control" />
<input type="button" id="sensor_query" value="查询" /> (触发器 ID 为空时查询所有触
发器信息，输入 ID 后查询指定触发器信息)
</div>
<hr/>
<!--创建传感器类型触发器-->
<div class="create block">
<label for="sensor_name">触发器名</label>
<input type="text" id="sensor_name" class="form-control"/>
<label for="sensor_type">触发器类型</label>
<input type="text" id="sensor_type" class="form-control" readonly="readonly"
value="sensor"/>
<label for="sensor_mac">MAC 地址</label>
<input type="text" id="sensor_mac" class="form-control"/>
<label for="sensor_ch">通道</label>
<select class="form-control" id="sensor_ch">
<option value="A0">A0</option>
<option value="A1">A1</option>
<option value="A2">A2</option>
<option value="A3">A3</option>
```




```

<option value="A4">A4</option>
<option value="A5">A5</option>
<option value="A6">A6</option>
<option value="D0">D0</option>
<option value="D1">D1</option>
<option value="D2">D2</option>
<option value="D3">D3</option>
<option value="D4">D4</option>
</select>
<label for="sensor_op">计算</label>
<select class="form-control" id="sensor_op">
<option value="">>>大于</option>
<option value=">=">大于或等于</option>
<option value="<">小于</option>
<option value="<=">小于或等于</option>
<option value="==">等于</option>
<option value="!=">不等于</option>
<option value="CHANGE">CHANGE</option>
<option value="&">与</option>
<option value="!&">与后去反</option>
</select>
<label for="sensor_value">值</label>
<input type="text" id="sensor_value" class="form-control"/>
<label for="sensor_once">触发规则</label>
<select class="form-control" id="sensor_once">
<option value="true">第一次触发</option>
<option value="false">每次触发</option>
</select>
<input type="button" id="sensor_create" class="form_button" value="创建传感器
类型触发器" />
</div>
<hr/>
<!--创建定时器类型触发器-->
<div class="create_block">
<label for="sensor_name">触发器名</label>
<input type="text" id="timer_name" class="form-control"/>
<label for="sensor_type">触发器类型</label>
<input type="text" id="timer_type" class="form-control" readonly="readonly"
value="timer"/>

<label for="timer_year">年</label>
<input type="text" id="timer_year" class="form-control"/>
<label for="timer_nonth">月</label>
<input type="text" id="timer_nonth" class="form-control"/>
<label for="timer_week">周</label>
<input type="text" id="timer_week" class="form-control"/>
<label for="timer_day_of_week">周几</label>
<input type="text" id="timer_day_of_week" class="form-control"/>
<label for="timer_day">日</label>
<input type="text" id="timer_day" class="form-control"/>

```



```

<label for="timer_hour">时</label>
<input type="text" id="timer_hour" class="form-control"/>
<label for="timer_minute">分</label>
<input type="text" id="timer_minute" class="form-control"/>
<label for="timer_second">秒</label>
<input type="text" id="timer_second" class="form-control"/>
<input type="button" id="timer_create" class="form_button" value="创建定时器
类型触发器" />
    (创建定时器类型触发器是，值为"/7"格式，表示每隔多久触发。)
</div>
<hr/>

```

```

<div id="data_show" class="data block">
<!--显示操作后数据-->
</div>

```

③ 编写 css 样式。为了使得 auto_sensor.html 排版更加美观，需要编写 css 样式，根据第②步骤编写的 HTML 标签及 auto_sensor.html 参考页面的设计图，在<head></head>中添加如下 css 样式代码。

```

<style>
h2{
    text-align:center;
}
hr{
    width:1024px;
    margin:0px auto;
    border: 1px dashed #666;
}
label{
    width: 80px;
}
.form-control{
    width: 80px;
    margin-top:10px;
}
.form_button{
    margin-top:10px;
}
.server .form-control{
    width: 200px;
}
.block{
    width:1024px;
    margin:10px auto;
}
</style>

```

上述步骤完成后，便实现了触发器的 Web 页面编写，显示效果如图 2.52 所示。



自动控制触发器演示

应用 ID

1155223953

密钥

Xrk6UicNrbo3KtYDDaUg9HAMH

服务器地址

zhiyun360.com:8001

触发器ID

查询

(触发器ID为空时查询所有触发器信息，输入ID后查询指定触发器信息)

触发器名

触发器类型

sensor

mac地址

通道

A0

计算

大于

值

触发规则

第一次触

创建传感器类型触发器

触发器名

触发器类型

timer

年

月

周

周几

日

时

分

秒

创建定时器类型触发器

(创建定时器类型触发器是，值为“*/?”格式，表示每隔多久触发。)

图 2.52 触发器 Web 页面显示

(4) Web 逻辑代码实现。

① 引入 js 脚本库。添加“jquery-1.11.0.min.js”和“WSNAutoctrl.js”脚本库文件，添加方法为在<head></head>标签中添加如下代码。

```
<script src="../../WSN/jquery-1.11.0.min.js"></script>
<script src="../../WSN/WSNAutoctrl.js"></script>
```

说明：“../”为进入上一级目录。

② 编写 js 逻辑代码。

```
<script>
//初始化查询操作
ac = new WSNAutoctrl();

$(function(){
    //查询触发器
    $("#sensor_query").click(function(){
        var aid = $("#aid").val();
        var xkey = $("#xkey").val();
        var saddr = $("#saddr").val();
        ac.setIdKey(aid,xkey);           //设置链接参数
        ac.setServerAddr(saddr);       //设置服务器地址
        var num = $("#sensor_id").val(); //设置查询 ID
        if(isNaN(parseInt(num))){       //传递字符串不能转换为数字查询所有 ID
            num = "";
        }else{                          //传递字符串能转换为数字查询指定 ID
            num = parseInt(num);
        }
        ac.getTrigger(num, function(dat){
            var data = JSON.stringify(dat); //json 对象变为字符串
            $("#data_show").text(data);
        });
    });
});

//创建传感器类型触发器
```




```

$("#sensor_create").click(function(){
    var aid = $("#aid").val();
    var xkey = $("#xkey").val();
    var saddr = $("#saddr").val();
    ac.setIdKey(aid,xkey);           //设置链接参数
    ac.setServerAddr(saddr);        //设置服务器地址
    var name = $("#sensor_name").val();
    var type = $("#sensor_type").val();
    var pa = {};
    pa["mac"] = $("#sensor_mac").val();
    pa["ch"] = $("#sensor_ch").val();
    pa["op"] = $("#sensor_op").val();
    pa["value"] = $("#sensor_value").val();
    var once = $("#sensor_once").val();
    //once 值为布尔类型
    if(once=="true"){
        once =true;
    }else{
        once =false;
    }
    pa["once"] = once;
    ac.createTrigger(name, type, pa, function(dat){
        //var data = JSON.stringify(dat);    //json 对象变为字符串
        $("#data_show").text(data);
    });
});

```

//创建定时器类型触发器

```

$("#timer_create").click(function(){
    var aid = $("#aid").val();
    var xkey = $("#xkey").val();
    var saddr = $("#saddr").val();
    ac.setIdKey(aid,xkey);           //设置链接参数
    ac.setServerAddr(saddr);        //设置服务器地址
    var name = $("#timer_name").val();
    var type = $("#timer_type").val();
    var pa = {};
    pa["year"] = $("#timer_year").val();
    pa["month"] = $("#timer_month").val();
    pa["week"] = $("#timer_week").val();
    pa["day_of_week"] = $("#timer_day_of_week").val();
    pa["hour"] = $("#timer_hour").val();
    pa["minute"] = $("#timer_minute").val();
    pa["second"] = $("#timer_second").val();
    ac.createTrigger(name, type, pa, function(dat){
        var data = JSON.stringify(dat);    //json 对象变为字符串
        $("#data_show").text(data);
    });
});

```

```

})
</script>

```



(5) 执行器。在本任务中 auto_actuator.html 页面的样式设计如下图所示,第1行为自动控应用 ID、应用 KEY、服务器地址配置,第2栏执行器查询栏,第3栏为执行器(传感器类型)创建栏,第4栏为执行器(摄像头类型)创建栏,第5栏为执行器(电话类型)创建栏,第6栏为执行器(任务类型)创建栏,第7栏为查询执行器显示栏,如图 2.53 所示。

自动控制触发器演示

应用ID

1155223953

密钥

Xrk6UicNrbo3KiX1tYDDaUq9HAMF

服务器地址

t.zhiyun360.com:8001

执行器ID

查询

(执行器ID为空时查询所有执行器信息,输入ID后查询指定执行器信息)

执行器名

执行器类型

sensor

mac地址

指令

创建传感器类型执行器

执行器名

执行器类型

ipcamera

摄像头地址

用户名

密码

指令

创建传感器类型执行器

执行器名

执行器类型

phone

mac地址

指令

创建传感器类型执行器

执行器名

执行器类型

job

任务ID

使能

使能

创建传感器类型执行器

图 2.53 auto_actuator.html 页面的样式设计

(6) Web 页面样式实现。

① 构建 HTML 页面。在 Web/examples 目录下新建一个项目文件夹,命名为 autoctrl,然后打开记事本,在记事本中输入以下 HTML 语句,输入完后保存为 utf-8 格式,文件命名为 auto_actuator.html,并将该文件保存到 Web/examples/autoctrl 文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>自动控制执行器</title>
</head>
<body>
</body>
</html>
```

② 添加 HTML 标签。在<body></body>标签中添加 HTML 标签,内容如下所示。

```
<h2>自动控制触发器演示</h2>
<!--配置服务器信息-->
<div class="server block">
<label for="aid">应用 ID</label>
<input type="text" id="aid" value="123"/>
<label for="xkey">密钥</label>
<input type="text" id="xkey" class="form-control" value="123" />
<label for="saddr">服务器地址</label>
<input type="url" id="saddr" class="form-control" value="zhiyun360.com:8001"/>
</div>
<hr/>
<!--查询执行器-->
```



```

<div class="query block">
<label for="actuator_id">执行器 ID</label>
<input type="text" id="actuator_id" class="form-control" />
<input type="button" id="actuator_query" value="查询" />(执行器 ID 为空时查询所有执行器信息, 输入 ID 后查询指定执行器信息)
</div>
<hr/>
<!--创建传感器类型执行器-->
<div class="create block">
<label for="sensor_name">执行器名</label>
<input type="text" id="sensor_name" class="form-control"/>
<label for="sensor_type">执行器类型</label>
<input type="text" id="sensor_type" class="form-control" readonly="readonly"
value="sensor"/>
<label for="sensor_mac">MAC 地址</label>
<input type="text" id="sensor_mac" class="form-control"/>
<label for="sensor_data">指令</label>
<input type="text" id="sensor_data" class="form-control"/>
<input type="button" id="sensor_create" class="form_button" value="创建传感器
类型执行器" />
</div>
<hr/>
<!--创建摄像头类型执行器-->
<div class="create block">
<label for="cam_name">执行器名</label>
<input type="text" id="cam_name" class="form-control"/>
<label for="cam_type">执行器类型</label>
<input type="text" id="cam_type" class="form-control" readonly="readonly"
value="ipcamera"/>
<label for="cam_mac">摄像头地址</label>
<input type="text" id="cam_mac" class="form-control"/>
<label for="cam_user">用户名</label>
<input type="text" id="cam_user" class="form-control"/>
<label for="cam_pwd">密码</label>
<input type="text" id="cam_pwd" class="form-control"/>
<label for="cam_data">指令</label>
<input type="text" id="cam_data" class="form-control"/>
<input type="button" id="cam_create" class="form_button" value="创建传感器类
型执行器" />
</div>
<hr/>
<!--创建短信类型执行器-->
<div class="create block">
<label for="phone_name">执行器名</label>
<input type="text" id="phone_name" class="form-control"/>
<label for="phone_type">执行器类型</label>
<input type="text" id="phone_type" class="form-control" readonly="readonly"
value="phone"/>
<label for="phone_mac">MAC 地址</label>
<input type="text" id="phone_mac" class="form-control"/>

```




```

<label for="phone_data">指令</label>
<input type="text" id="phone_data" class="form-control"/>
<input type="button" id="phone_create" class="form_button" value="创建传感器
类型执行器" />
</div>
<hr/>
<!--创建任务类型执行器-->
<div class="create block">
<label for="job_name">执行器名</label>
<input type="text" id="job_name" class="form-control"/>
<label for="job_type">执行器类型</label>
<input type="text" id="job_type" class="form-control" readonly="readonly"
value="job"/>
<label for="job_jid">任务 ID</label>
<input type="text" id="job_jid" class="form-control"/>
<label for="job_enable">使能</label>
<select class="form-control" id="job_enable">
<option value="true">使能</option>
<option value="false">禁止</option>
</select>
<input type="button" id="job_create" class="form_button" value="创建传感器类
型执行器" />
</div>
<hr/>
<div id="data_show" class="data block">
<!--显示操作后数据-->
</div>

```

③ 编写 css 样式。为了使得 auto_actuator.html 排版更加美观,需要编写 css 样式,根据第②步骤编写的 HTML 标签及 auto_actuator.html 参考页面的设计图,在<head></head>中添加如下 css 样式代码。

```

<style>
h2{
    text-align:center;
}
hr{
    width:1024px;
    margin:0px auto;
    border: 1px dashed #666;
}
label{
    width: 80px;
}
.form-control{
    width: 80px;
    margin-top:10px;
}
.form_button{
    margin-top:10px;
}

```



```
.server .form-control{
    width: 200px;
}
.block{
    width:1024px;
    margin:10px auto;
}
</style>
```

(7) Web 逻辑代码实现。

① 引入 js 脚本库。添加 jquery-1.11.0.min.js 和 WSNAutoctrl.js 脚本库文件，添加方法为在<head></head>标签中添加如下代码。

```
<script src="../../WSN/jquery-1.11.0.min.js"></script>
<script src="../../WSN/WSNAutoctrl.js"></script>
```

说明：“../”为进入上一级目录。

② 编写 js 逻辑代码。

```
<script>
//初始化查询操作
ac = new WSNAutoctrl();

$(function(){
    //查询执行器
    $("#actuator_query").click(function(){
        var aid = $("#aid").val();
        var xkey = $("#xkey").val();
        var saddr = $("#saddr").val();
        ac.setIdKey(aid,xkey);           //设置链接参数
        ac.setServerAddr(saddr);        //设置服务器地址
        var num = $("#actuator_id").val(); //设置查询 ID
        if(isNaN(parseInt(num))){        //传递字符串不能转换为数字查询所有 ID
            num = "";
        }else{                          //传递字符串能转换为数字查询指定 ID
            num = parseInt(num);
        }
        ac.getActuator(num, function(dat){
            var data = JSON.stringify(dat); //json 对象变为字符串
            $("#data_show").text(data);
        });
    });

    //创建传感器类型执行器
    $("#sensor_create").click(function(){
        var aid = $("#aid").val();
        var xkey = $("#xkey").val();
        var saddr = $("#saddr").val();
        ac.setIdKey(aid,xkey);           //设置链接参数
        ac.setServerAddr(saddr);        //设置服务器地址
        var name = $("#sensor_name").val();
        var type = $("#sensor_type").val();
        var pa = {};
```



```
pa["mac"] = $("#sensor_mac").val();
pa["data"] = $("#sensor_data").val();
ac.createActuator(name, type, pa, function(dat){
    var data = JSON.stringify(dat);        //json 对象变为字符串
    $("#data_show").text(data);
});
});

//创建摄像头类型执行器
$("#timer_create").click(function(){
    var aid = $("#aid").val();
    var xkey = $("#xkey").val();
    var saddr = $("#saddr").val();
    ac.setIdKey(aid,xkey);                  //设置链接参数
    ac.setServerAddr(saddr);              //设置服务器地址
    var name = $("#cam_name").val();
    var type = $("#cam_type").val();
    var pa = {};
    pa["mac"] = $("#cam_mac").val();
    pa["user"] = $("#cam_user").val();
    pa["pwd"] = $("#cam_pwd").val();
    pa["data"] = $("#cam_data").val();
    ac.createActuator(name, type, pa, function(dat){
        var data = JSON.stringify(dat);    //json 对象变为字符串
        $("#data_show").text(data);
    });
});

//创建短信类型执行器
$("#phone_create").click(function(){
    var aid = $("#aid").val();
    var xkey = $("#xkey").val();
    var saddr = $("#saddr").val();
    ac.setIdKey(aid,xkey);                  //设置链接参数
    ac.setServerAddr(saddr);              //设置服务器地址
    var name = $("#phone_name").val();
    var type = $("#phone_type").val();
    var pa = {};
    pa["mac"] = $("#phone_mac").val();
    pa["data"] = $("#phone_data").val();
    ac.createActuator(name, type, pa, function(dat){
        var data = JSON.stringify(dat);    //json 对象变为字符串
        $("#data_show").text(data);
    });
});

//创建任务类型执行器
$("#job_create").click(function(){
    var aid = $("#aid").val();
    var xkey = $("#xkey").val();
    var saddr = $("#saddr").val();
    ac.setIdKey(aid,xkey);                  //设置链接参数
```




```
ac.setServerAddr(saddr);           //设置服务器地址
var name = $("#job_name").val();
var type = $("#job_type").val();
var pa = {};
pa["jid"] = $("#job_jid").val();
var enable = $("#job_enable").val();
//once 值为布尔类型
if(enable=="true"){
    enable =true;
}else{
    enable =false;
}
pa["enable"] = enable;
ac.createActuator(name, type, pa, function(dat){
    var data = JSON.stringify(dat);    //json 对象变为字符串
    $("#data_show").text(data);
});
});
})
</script>
```

(8) 执行任务。在本任务中 `auto_job.html` 页面的样式设计如下图所示, 第 1 行为自动控应用 ID、应用 KEY、服务器地址配置, 第 2 栏执行任务查询栏, 第 3 栏为执行任务创建栏, 第 4 栏为查询执行任务显示栏, 如图 2.54 所示。

自动控制执行任务演示			
应用ID	<input type="text" value="1155223953"/>	密钥	<input type="text" value="Xrk6UichNrbo3Kix1tYDDaUq9HAMH"/>
服务器地址	<input type="text" value="t.zhiyun360.com:8001"/>		
<hr/>			
执行任务ID	<input type="text"/>	<input type="button" value="查询"/> (执行任务ID为空时查询所有执行任务信息, 输入ID后查询指定执行任务信息)	
<hr/>			
任务名	<input type="text"/>	任务使能	<input type="button" value="使能"/>
触发器ID	<input type="text"/>	执行器ID	<input type="text"/>
<input type="button" value="创建任务"/>			
<hr/>			

图 2.54 自动控制执行任务页面设计

(9) Web 页面样式实现。

① 构建 HTML 页面。在 `Web/examples` 目录下新建一个项目文件夹, 命名为 `autoctrl`, 然后打开记事本, 在记事本中输入以下 HTML 语句, 输入完后保存为 `utf-8` 格式, 文件命名为 `auto_job.html`, 并将该文件保存到 “`Web/examples/autoctrl`” 文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>自动控制执行任务</title>
</head>
<body>
</body>
</html>
```



② 添加 HTML 标签。在<body></body>标签中添加 HTML 标签，内容如下所示。

```
<h2>自动控制执行任务演示</h2>
<!--配置服务器信息-->
<div class="server block">
<label for="aid">应用 ID</label>
<input type="text" id="aid" value="123"/>
<label for="xkey">密钥</label>
<input type="text" id="xkey" class="form-control" value="123" />
<label for="saddr">服务器地址</label>
<input type="url" id="saddr" class="form-control" value="zhiyun360.com:8001"/>
</div>
<hr/>
<!--查询执行任务-->
<div class="query block">
<label for="job_id">执行任务 ID</label>
<input type="text" id="job_id" class="form-control" />
<input type="button" id="job_query" value="查询" /> (执行任务 ID 为空时查询所有执行任务信息，输入 ID 后查询指定执行任务信息)
</div>
<hr/>
<!--创建执行任务-->
<div class="create block">
<label for="sensor_name">任务名</label>
<input type="text" id="job_name" class="form-control"/>
<label for="job_enable">任务使能</label>
<select class="form-control" id="job_enable">
<option value="true">使能</option>
<option value="false">禁止</option>
</select>
<label for="job_tids">触发器 ID</label>
<input type="text" id="job_tids" class="form-control"/>
<label for="job_aids">执行器 ID</label>
<input type="text" id="job_aids" class="form-control"/>
<input type="button" id="job_create" class="form_button" value="创建任务" />
</div>
<hr/>
<div id="data_show" class="data block">
<!--显示操作后数据-->
</div>
```

③ 编写 css 样式。为了使得 auto_job.html 排版更加美观，需要编写 css 样式，根据第②步骤编写的 HTML 标签及 auto_job.html 参考页面的设计图，在<head></head>中添加如下 css 样式代码。

```
<style>
h2{
    text-align:center;
}
hr{
    width:1024px;
    margin:0px auto;
```



```
border: 1px dashed #666;
}
label{
width: 80px;
}
.form-control{
width: 80px;
margin-top:10px;
}
.form_button{
margin-top:10px;
}
.server .form-control{
width: 200px;
}
.block{
width:1024px;
margin:10px auto;
}
</style>
```

(10) Web 逻辑代码实现

① 引入 js 脚本库。添加 jquery-1.11.0.min.js 和 WSNAutoctrl.js 脚本库文件，添加方法为在<head></head>标签中添加如下代码。

```
<script src="../../WSN/jquery-1.11.0.min.js"></script>
<script src="../../WSN/WSNAutoctrl.js"></script>
```

说明：“../”为进入上一级目录。

② 编写 js 逻辑代码。

```
<script>
//初始化查询操作
ac = new WSNAutoctrl();

$(function(){
//查询任务
$("#job_query").click(function(){
var aid = $("#aid").val();
var xkey = $("#xkey").val();
var saddr = $("#saddr").val();
ac.setIdKey(aid,xkey);           //设置链接参数
ac.setServerAddr(saddr);       //设置服务器地址
var num = $("#job_id").val();   //设置查询 ID
if(isNaN(parseInt(num))) {      //传递字符串不能转换为数字查询所有 ID
num = "";
} else {                        //传递字符串能转换为数字查询指定 ID
num = parseInt(num);
}
ac.getJob(num, function(dat){
var data = JSON.stringify(dat); //json 对象变为字符串
$("#data_show").text(data);
});
});
```




```

    })

    //创建任务
    $("#job_create").click(function(){
        var aid = $("#aid").val();
        var xkey = $("#xkey").val();
        var saddr = $("#saddr").val();
        ac.setIdKey(aid,xkey);
        ac.setServerAddr(saddr);
        var name = $("#job_name").val();
        var enable = $("#job_enable").val();
        //once 值为布尔类型
        if(enable=="true"){
            enable =true;
        }else{
            enable =false;
        }
        var pa = {
            tids:[],
            aids:[]
        };
        var tids = $("#job_tids").val();
        var aids = $("#job_aids").val();
        var t = tids.split(",");
        var a = aids.split(",");
        for(var i=0;i<t.length;i++){
            pa.tids[i] = parseInt(t[i]);
        }
        for(var i=0;i<a.length;i++){
            pa.aids[i] = parseInt(a[i]);
        }
        ac.createJob(name, enable, pa, function(dat){
            var data = JSON.stringify(dat);
            $("#data_show").text(data);
        });
    });
    })
</script>
```

(11) 执行历史记录。在本任务中 auto_scheduduler.html 页面的样式设计如图 2.55 所示，第 1 行为自动控应用 ID、应用 KEY、服务器地址配置，第 2 栏执行记录查询栏，第 3 栏为查询执行记录显示栏。

自动控制执行记录演示

应用ID

1155223953

密钥

Xrk6UicNrbo3KIX1tYDDaUq9HAMH

服务器地址

1.zhiyun360.com:8001

执行记录ID

查询

(执行任务ID为空时查询所有执行任务信息，输入ID后查询指定执行任务信息)

图 2.55 自动控制执行记录页面设计



(12) Web 页面样式实现。

① 构建 HTML 页面。在“Web/examples”目录下新建一个项目文件夹，命名为 autoctrl，然后打开记事本，在记事本中输入以下 HTML 语句，输入完后保存为 utf-8 格式，文件命名为“auto_schedudler.html”，并将该文件保存到“Web/examples/autoctrl”文件夹中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>自动控制执行记录</title>
</head>
<body>
</body>
</html>
```

② 添加 HTML 标签。在<body></body>标签中添加 HTML 标签，内容如下所示。

```
<h2>自动控制执行记录演示</h2>
<!--配置服务器信息-->
<div class="server block">
<label for="aid">应用 ID</label>
<input type="text" id="aid" value="123"/>
<label for="xkey">密钥</label>
<input type="text" id="xkey" class="form-control" value="123" />
<label for="saddr">服务器地址</label>
<input type="url" id="saddr" class="form-control" value="zhiyun360.com:8001"/>
</div>
<hr/>
<!--查询执行任务-->
<div class="query block">
<label for="job_id">执行记录 ID</label>
<input type="text" id="sc_id" class="form-control" />
<input type="button" id="sc_query" value="查询" /> (执行任务 ID 为空时查询所有执行
任务信息，输入 ID 后查询指定执行任务信息)
</div>
<hr/>
<div id="data_show" class="data block">
<!--显示操作后数据-->
</div>
```

③ 编写 css 样式。为了使得 auto_schedudler.html 排版更加美观，需要编写 css 样式，根据第②步骤编写的 HTML 标签及 auto_schedudler.html 参考页面的设计图，在<head></head>中添加如下 css 样式代码。

```
<style>
h2{
    text-align:center;
}
hr{
    width:1024px;
    margin:0px auto;
    border: 1px dashed #666;
```



```
}  
label{  
    width: 80px;  
}  
.form-control{  
    width: 80px;  
    margin-top:10px;  
}  
.form_button{  
    margin-top:10px;  
}  
.server .form-control{  
    width: 200px;  
}  
.block{  
    width:1024px;  
    margin:10px auto;  
}  
</style>
```

(13) Web 逻辑代码实现。

① 引入 js 脚本库。添加 jquery-1.11.0.min.js 和 WSNAutoctrl.js 脚本库文件，添加方法为在<head></head>标签中添加如下代码。

```
<script src="../../../WSN/jquery-1.11.0.min.js"></script>  
<script src="../../../WSN/WSNAutoctrl.js"></script>
```

说明：“../”为进入上一级目录。

② 编写 js 逻辑代码。

```
<script>  
//初始化查询操作  
ac = new WSNAutoctrl();  
  
$(function(){  
    //查询执行记录  
    $("#sc_query").click(function(){  
        var aid = $("#aid").val();  
        var xkey = $("#xkey").val();  
        var saddr = $("#saddr").val();  
        ac.setIdKey(aid,xkey);           //设置链接参数  
        ac.setServerAddr(saddr);        //设置服务器地址  
        var num = $("#sc_id").val();     //设置查询 ID  
        if(isNaN(parseInt(num))){        //传递字符串不能转换为数字查询所有 ID  
            num = "";  
        }else{//传递字符串能转换为数字查询指定 ID  
            num = parseInt(num);  
        }  
        ac.getJob(num, function(dat){  
            var data = JSON.stringify(dat); //json 对象变为字符串  
            $("#data_show").text(data);  
        });  
    });  
})
```




```
})
</script>
```

2.5.5 开发步骤

1. 曲线的实现

(1) 在“Web\examples”目录下新建一个项目文件夹，命名为 **curve**，然后打开记事本，在记事本中输入以下 HTML 语句。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>曲线图的实现</title>
</head>
<body>
</body>
</html>
```

输入完后保存，文件命名为 **curve** 并以 .html 格式后缀结尾，并将该文件保存到 **curve** 文件夹中。

说明：`<!DOCTYPE html.....>`与`<html xmlns="http://www.w3.org/1999/xhtml">`标签内容为标准规范，在 html 中建议用户添加这两个标签内容，否则可能会导致意想不到的错误。

`<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />`标签的作用是支持文本的编码格式，如果不添加的话，可能会导致 js 文件中的汉字显示乱码。由于在这个页面中没有添加任何内容，所以运行这个页面后，将会显示空白。

(2) 在 HTML 中添加 jQuery 语言库 `jquery-1.11.0.min.js` 和曲线图实现的 `highcharts.js` 库的引用，然后添加曲线图控件绘制 API 的 `drawcharts.js` 文件，添加方法：在`<head></head>`标签中添加如下代码。

```
<script src="../../../js/jquery-1.11.0.min.js" type="text/javascript"></script>
<script src="../../../js/highcharts.js" type="text/javascript"></script>
<script src="../../../js/drawcharts.js" type="text/javascript"></script>
```

说明：“../”为进入上一级目录。

(3) 在`<body></body>`标签中添加 HTML 标签，内容如下。

```
<p>在此处显示曲线图:</p>
<div id="curve" >
</div>
```

(4) 编写 js 脚本调用曲线绘制的 API，在`<head></head>`标签中添加如下 js 代码。

```
<script type="text/javascript">
$(function(){
    //曲线图显示用的二维数组数据
    var data = [[1398368037823,2],[1398470377015,6],
    [1398556786135,1],[1398643177964,9],
    [1398710239656,10],[1398784852105,7]];
    showChart('#curve', 'spline', '', false,data); //画曲线
});
</script>
```



说明:

- `$(function(){}))`为文档就绪函数（也可叫入口函数），功能是当所有的 HTML 标签加载完毕之后开始执行此方法内编写的 js 代码。
- `var data` 声明的是一个二维数组，以`[1398368037823,2]`为例进行说明，第一个元素是代表横坐标，为距离 1970-01-01 的毫秒数，第二个元素是纵坐标，为需要显示的值。
- `showChart('#curve', 'spline', '', false, eval(data))`方法是在 `drawcharts.js` 中定义的，参数说明：第一个参数为曲线图在 HTML 标签中显示的 ID，第二个参数为曲线显示的类型，`spline` 为平滑，第三个参数为数据的显示单位（本例程中为空字符），第四个参数默认为 `false`，第五个参数即为曲线图的数据来源（注：必须是二维数组）。

综合上述的说明不难发现此函数入口的功能就是在 `curve.html` 中 `id="curver"` 的标签中画曲线类型为 `spline`，单位为 `℃`，数据源为 `data` 数组的曲线图。

(5) 保存 `curve.html`，然后双击 `curve.html` 文件访问该页面，显示效果，如图 2.56 所示。

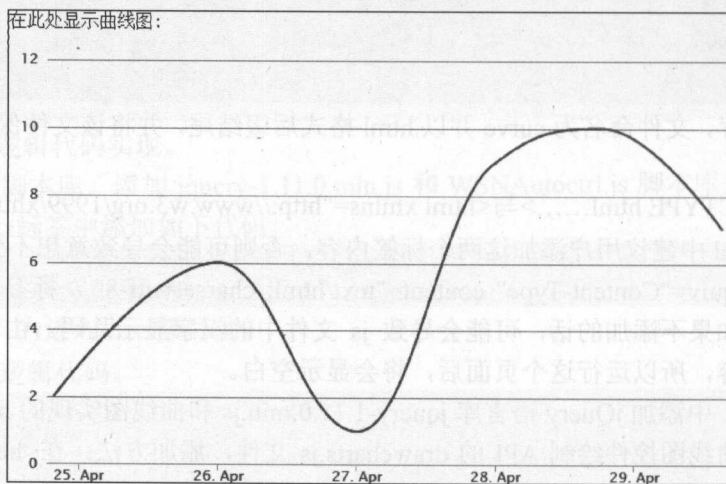


图 2.56 曲线图页面显示

2. 仪表的实现

(1) 在 `Web\examples` 目录下新建一个项目文件夹，命名为 `dial`，然后打开记事本，在记事本中输入以下 HTML 语句。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>表盘的实现</title>
</head>
<body>
</body>
</html>
```

输入完后保存，文件命名为 `dial` 并以 `.html` 格式后缀结尾，并将该文件保存到 `dial` 文件夹中。

说明：`<!DOCTYPE html.....>`与`<html xmlns="http://www.w3.org/1999/xhtml">`标签内容为



标准规范, 在 HTML 中建议用户添加这两个标签内容, 否则可能会导致意想不到的错误。

`<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />`标签的作用是支持文本的编码格式, 如果不添加的话, 可能会导致 js 文件中的汉字显示乱码。

由于在这个页面中没有添加任何内容, 所以运行这个页面后, 将会显示空白。

(2) 在 HTML 中添加 jQuery 语言库 `jquery-1.11.0.min.js` 和表盘实现的 `highcharts.js` 和 `highcharts-more.js` 库的引用, 然后添加表盘控件绘制 API 的 `drawcharts.js` 文件, 添加方法: 在 `<head></head>` 标签中添加如下代码。

```
<script src="../../js/jquery-1.11.0.min.js" type="text/javascript"></script>
<script src="../../js/highcharts.js" type="text/javascript"></script>
<script src="../../js/highcharts-more.js" type="text/javascript"></script>
<script src="../../js/drawcharts.js" type="text/javascript"></script>
```

说明: “../” 为进入上一级目录。

(3) 添加完 js 库的引用后, 开始让 html 页面中显示表盘, 在 `<body></body>` 标签中添加如下标签内容。

```
<p>在此处显示表盘:</p>
<div id="dial" >
</div>
```

然后添加用户自定义的脚本文件, 以实现表盘在 HTML 中的绘制, 在 `<head></head>` 标签中输入如下内容。

```
<script type="text/javascript">
$(function(){
    getDial("#dial", "", "温度", "°C", 0, 100, { layer1: { from: 10, to: 30, color:
green }, layer2: { from: 0, to: 10, color: yellow }, layer3: { from: 30, to: 100,
color: red } });
})
</script>
```

说明: `$(function){}` 为文档就绪函数, 这个函数里面调用了 `getDial` 表盘绘制函数, 参数说明: `#dial` 表示表盘在 `id=dial` 的标签中显示, 然后后面的 `{ layer1: { from: 10, to: 30, color: green }, layer2: { from: 0, to: 10, color: yellow }, layer3: { from: 30, to: 100, color: red } }` 表示将表盘分为三个层, 第一层数据显示范围为 10~30, 颜色为 green, 第二层数据显示范围为 0~10, 颜色为 yellow, 第三层数据显示范围为 30~100, 颜色为 red。

编写完后保存该文件, 双击 `dial.html` 文件访问该页面, 显示效果, 如图 2.57 所示。

(4) 第 3 步骤实现了在指定位置绘制表盘的功能, 现在需要做的是在 HTML 页面中的表盘显示数据, 表盘数据显示的方法为直接调用 `setDialData()` 函数对表盘赋值, 以实现表盘指针的旋转及显示框的显示。实现方法: 在第 3 步骤中的 js 代码后面添加如下内容并保存。

```
setDialData('#dial', 67);
```

(5) 保存后, 双击 `dial.html` 文件访问该页面, 可以看到该表盘的指针值变化了, 显示效果如图 2.58 所示。

3. 实时数据的接收与发送

js 代码编写完毕后, 保存该文件, 双击 `realTimeData.html` 文件运行, 在 Web 页面中看到如图 2.59 所示的页面。

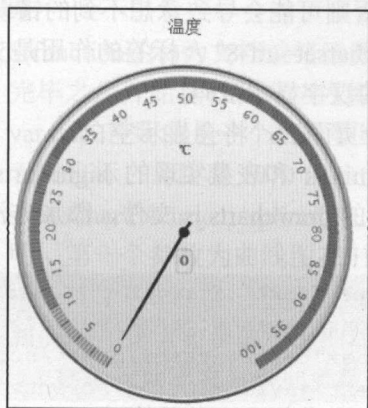


图 2.57 仪表显示界面

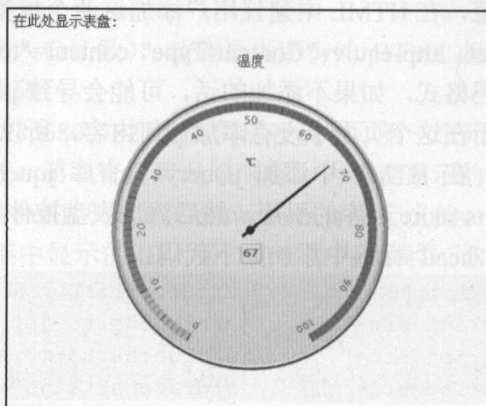


图 2.58 仪表显示界面

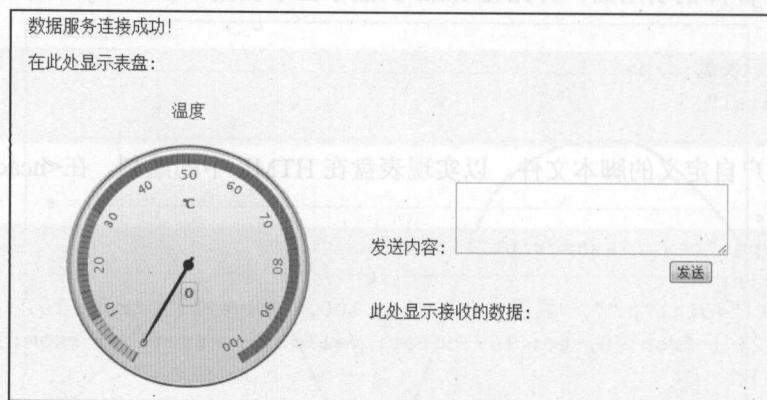


图 2.59 实时数据页面显示

页面显示“数据服务连接成功!”,表明数据服务已经成功连接,底层传感器与 Web 页面可以进行正常通信,在“发送内容”的输入框中输入“{A0=?}”(该命令为向底层的温湿度传感器查询当前温度值)命令,然后单击“发送”按钮,数据发送成功后便可在页面接收到从底层传过来的数据,同时将数据解析之后在表盘中显示,如图 2.60 所示。

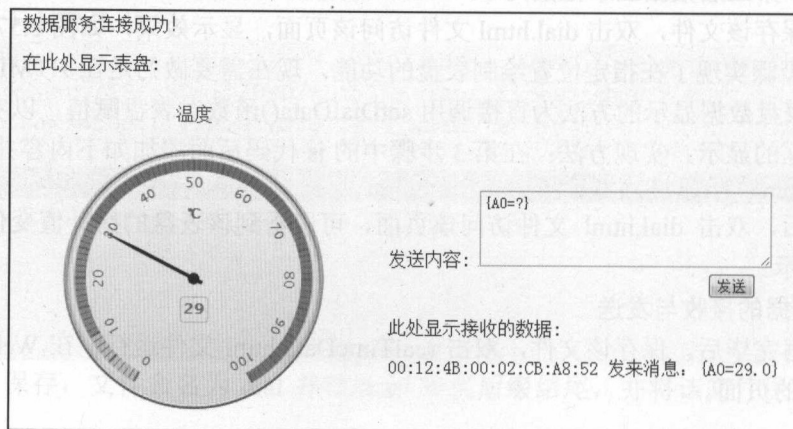


图 2.60 实时数据页面显示



从上图可得知，Web 端与底层传感器交互成功，Web 端发送 “{A0=?}” 查询命令后，底层传感器就发送 “{A0=29.0}” 数据至 Web 端。

4. 历史数据的获取与展示

js 代码编写完后保存 historyData.html，然后双击 historyData.html 文件访问该页面，将看到如图 2.61 所示的曲线图。

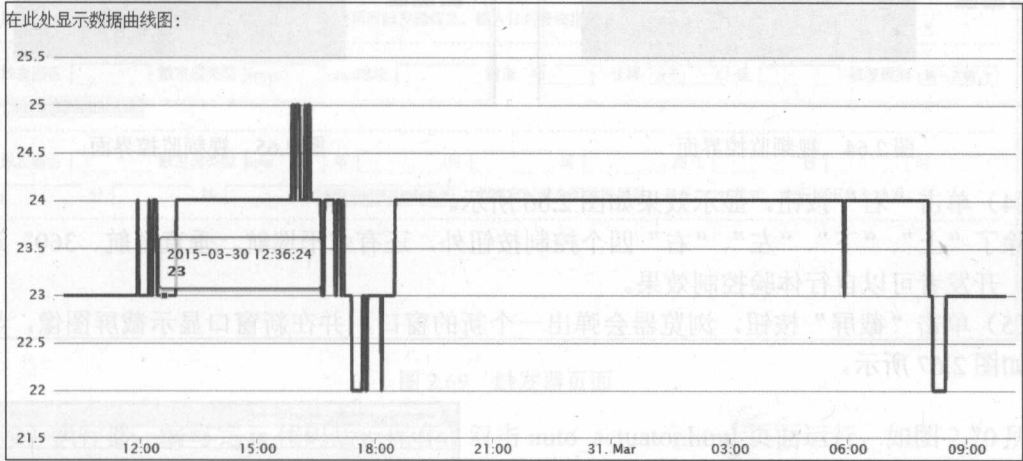


图 2.61 历史数据曲线图

5. 摄像头的显示与控制

编写完 js 代码后，保存，双击 WSNCamera.html 页面运行，然后单击 “开” 按钮，便可看到视频监控页面，如图 2.62 所示。

然后通过单击其他的控制按钮，摄像头就会执行相应的控制操作，同时视频监控画面也会实时的更新。

(1) 单击 “上” 按钮，显示效果如图 2.63 所示。

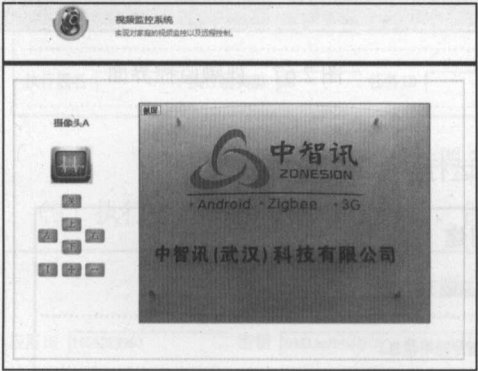


图 2.62 视频监控界面显示



图 2.63 视频监控界面

(2) 单击 “下” 按钮，显示效果如图 2.64 所示。

(3) 单击 “左” 按钮，显示效果如图 2.65 所示。



图 2.64 视频监控界面



图 2.65 视频监控界面

(4) 单击“右”按钮，显示效果如图 2.66 所示。

除了“上”、“下”、“左”、“右”四个控制按钮外，还有水平巡航、垂直巡航、360°巡航按钮，开发者可以自行体验控制效果。

(5) 单击“截屏”按钮，浏览器会弹出一个新的窗口，并在新窗口显示截屏图像，显示效果如图 2.67 所示。



图 2.66 视频监控界面



图 2.67 视频监控界面

6. 用户应用数据开发

编写完 js 代码后，保存，双击 property.html 页面运行，如图 2.68 所示。

应用数据查询与创建

应用ID

1155223953

密钥

Jxrk6UicNrbo3KIX1tYDDaUq9HAMH

服务器地址

11.zhiyun360.com:8080

应用名称

查询

(应用名称为空时查询所有应用信息，输入名称后查询指定应用信息)

应用名称

应用值

创建任务

图 2.68 应用数据页面



7. 自动控制模块开发

(1) 触发器。编写完 js 代码后，保存，双击 auto_sensor.html 页面运行，如图 2.69 所示。

自动控制触发器演示

应用ID

t155223953

密钥

Xrk6UicNrbo3KX1tYDDaUq9HAMH

服务器地址

t.zhiyun360.com:8001

触发器ID

查询

(触发器ID为空时查询所有触发器信息，输入ID后查询指定触发器信息)

触发器名

触发器类型

sensor

mac地址

通道

A0

计算

大于

值

触发规则

第一次触

创建传感器类型触发器

触发器名

触发器类型

timer

年

月

周

周几

日

时

分

秒

创建定时器类型触发器

(创建定时器类型触发器是，值为“*/7”格式，表示每隔多久触发。)

图 2.69 触发器页面

(2) 执行器。编写完 js 代码后，保存，双击 auto_actuator.html 页面运行，如图 2.70 所示。

自动控制触发器演示

应用ID

t155223953

密钥

Xrk6UicNrbo3KX1tYDDaUq9HAMH

服务器地址

t.zhiyun360.com:8001

执行器ID

查询

(执行器ID为空时查询所有执行器信息，输入ID后查询指定执行器信息)

执行器名

执行器类型

sensor

mac地址

指令

创建传感器类型执行器

执行器名

执行器类型

ipcamera

摄像头地址

用户名

密码

指令

创建传感器类型执行器

执行器名

执行器类型

phone

mac地址

指令

创建传感器类型执行器

执行器名

执行器类型

job

任务ID

使能

使能

创建传感器类型执行器

图 2.70 执行器页面

(3) 执行任务。编写完 js 代码后，保存，双击 auto_job.html 页面运行，如图 2.71 所示。

自动控制执行任务演示

应用ID

t155223953

密钥

Xrk6UicNrbo3KX1tYDDaUq9HAMH

服务器地址

t.zhiyun360.com:8001

执行任务ID

查询

(执行任务ID为空时查询所有执行任务信息，输入ID后查询指定执行任务信息)

任务名

任务使能

使能

触发器ID

执行器ID

创建任务

图 2.71 自动控制执行任务页面



(4) 执行历史记录。编写完 js 代码后，保存，双击 auto_scheduduler.html 页面运行，如图 2.72 所示。

自动控制执行记录演示

应用ID

1155223953

密钥

Xrk6UicNrbo3KiXtYDDaUq9HAMh

服务器地址

zhiyun360.com:8001

执行记录ID

查询

(执行任务ID为空时查询所有执行任务信息，输入ID后查询指定执行任务信息)

图 2.72 自动控制执行记录页面

2.5.6 总结与拓展

该任务实现了 WebAPI 开发，开发者可以编写一个温湿度采集的应用，在主界面每隔 30 s 更新一次温湿度的值。

2.6 任务 10：开发调试工具

2.6.1 学习目标

- 掌握数据分析工具、自动控制工具和网络拓扑工具的功能；
- 能用调试工具辅助项目开发。

2.6.2 开发环境

硬件：温度传感器 1 个，声光报警传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 2 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

2.6.3 原理学习

为了方便开发者快速使用智云平台，提供了智云开发调试工具，能够跟踪应用数据包及学习 API 的运用，该工具采用 Web 静态页面方式提供，功能如图 2.73 所示，主要包含以下内容。

- 智云数据分析工具，支持设备数据包的采集、监控及指令控制，支持智云数据库的历史数据查询。
- 智云自动控制工具，支持自动控制单元触发器、执行器、执行策略、执行记录的调试。
- 智云网络拓扑工具，支持进行传感器网络拓扑分析，能够远程更新传感网络 PANID 和 Channel 等信息。



欢迎进入API测试页面!

首页 实时数据 历史数据 网络拓扑 视频监控 应用数据 自动控制

实时数据

实时数据推送与采集测试工具。通过消息推送接口，能够实时抓取项目上下行所有节点数据包，支持通过命令对节点进行操作，获取节点实时信息、控制节点状态等操作。

历史数据

历数值/图片性历史数据获取测试工具。能够接入数据中心数据库，对项目任意时间段历史数据进行获取，支持数据型数据曲线图展示、JSON数据格式展示，同时支持摄像头抓拍的照片在曲线时间轴展示。

网络拓扑

ZigBee协议模式下网络拓扑图分析工具。能够实时接收并解析ZigBee网络数据包，将接收的网络信息通过拓扑图的形式展示，通过颜色对不同节点类型进行区分，显示节点的IEEE地址。

视频监控

视频监控测试工具。支持对项目摄像头进行管理，能够实时获取摄像头采集的画面，并支持对摄像头云台进行控制，支持上、下、左、右水平巡航、垂直巡航等，同时支持截屏操作。

应用数据

用户应用数据存储与查询测试工具。通过用户数据库接口，支持在该项目下存取用户数据，以key-value键值对的形式保存到数据中心服务器，同时支持通过key获取到其对应的value数值。

自动控制

自动控制模块测试工具。通过内置的逻辑编辑器实现复杂的自动控制逻辑，包括触发器（传感器类型、定时器类型）、执行器（传感器类型、短信类型、摄像头类型、任务类型）、执行任务、执行记录四大模块，每个模块都具有查询、创建、删除功能。

图 2.73 智云开发调试工具

2.6.4 开发内容

1. 实时推送测试工具

实时数据推送演示：通过消息推送接口，能够实时抓取项目上下行所有节点数据包，支持通过命令对节点进行操作，获取节点实时信息、控制节点状态等操作，如图 2.74 所示。

实时数据推送演示

配置服务器地址

应用ID

填写智云账号

密码

填写智云密码

服务器地址

zhilun360.com:28080

填写智云数据中心地址及端口号

新升

点击连接/断开按钮，与服务器进行连接

数据推送与接收

地址

数据

发送

此处可向指定地址的无线节点发送控制指令

数据记录

所有数据

清空数据

MAC地址	信息	时间
00:12:4B:00:02:63:C0:CF	{A0=0}	2015/9/6 11:32:11
00:12:4B:00:02:37:E7:7A	{A0=550.1}	2015/9/6 11:32:8
00:12:4B:00:02:60:E3:A9	{A0=11.0}	2015/9/6 11:32:8
00:12:4B:00:03:A7:E1:17	{D1=0}	2015/9/6 11:32:7
00:12:4B:00:02:C8:A9:C7	{A0=0}	2015/9/6 11:32:6
00:12:4B:00:02:C8:A8:52	{A0=0}	2015/9/6 11:32:4
00:12:4B:00:02:63:3C:B7	{A0=0}	2015/9/6 11:32:4
00:12:4B:00:02:60:E3:C8	{A0=11.0}	2015/9/6 11:31:51
00:12:4B:00:02:63:3C:93		
00:12:4B:00:02:63:3E:B5		

显示在线的无线节点地址

实时显示接收到的数据包

图 2.74 实时数据推送演示

2. 历史数据测试工具

历数值/图片性历史数据获取测试工具：能够接入到数据中心数据库，获取任意时间段历史数据，支持数值型数据曲线图展示、JSON 数据格式展示，同时支持摄像头抓拍的照片在曲

131 | PAGE



线时间轴展示，如图 2.75 所示。

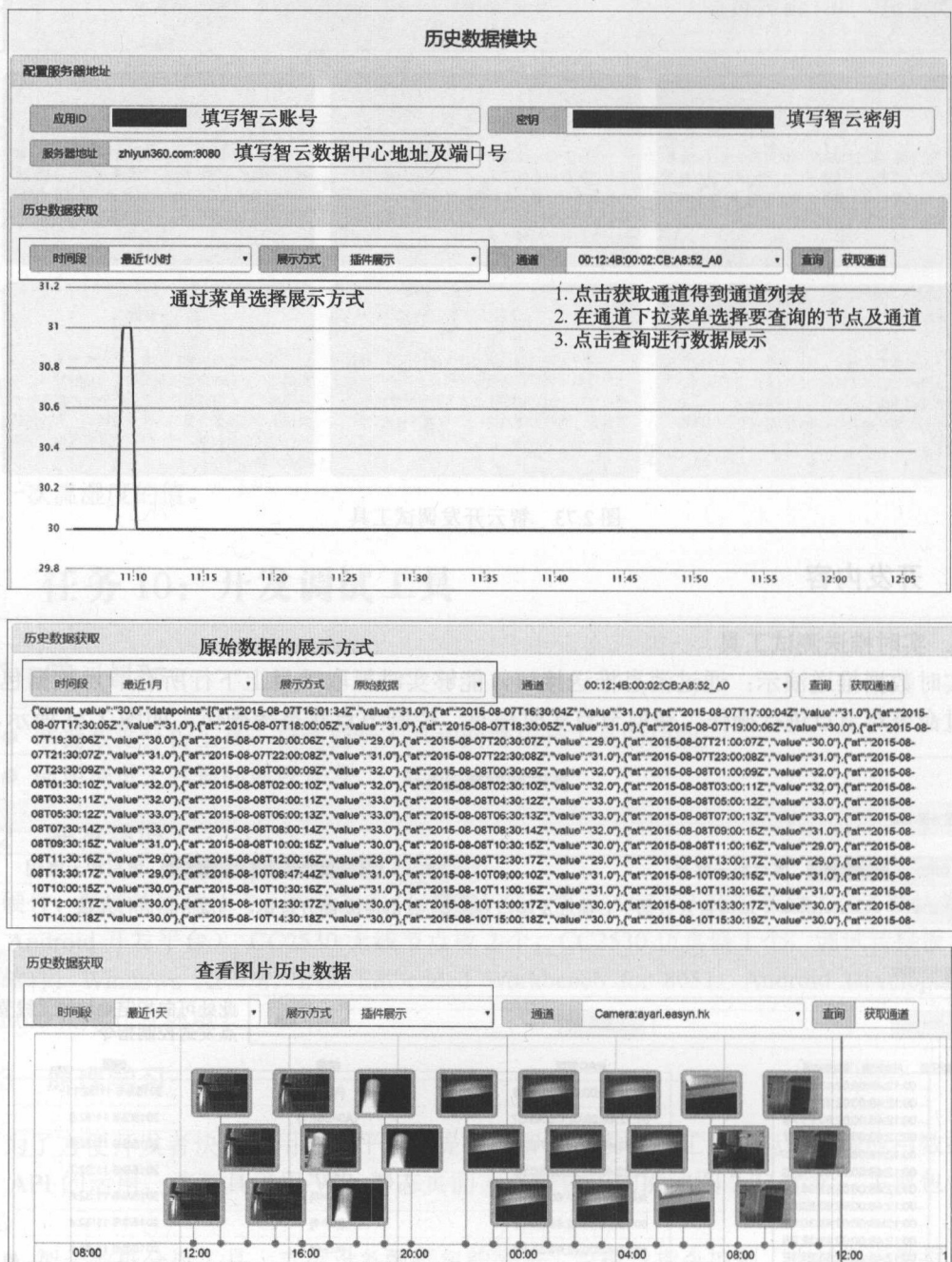


图 2.75 历史数据查询演示

3. 网络拓扑分析工具

ZigBee 网络拓扑图分析工具：能够实时接收并解析 ZigBee 网络数据包，将接收到的网络信息通过拓扑图的形式展示，通过颜色对不同节点类型进行区分，显示节点的 IEEE 地址，如图 2.76 所示。

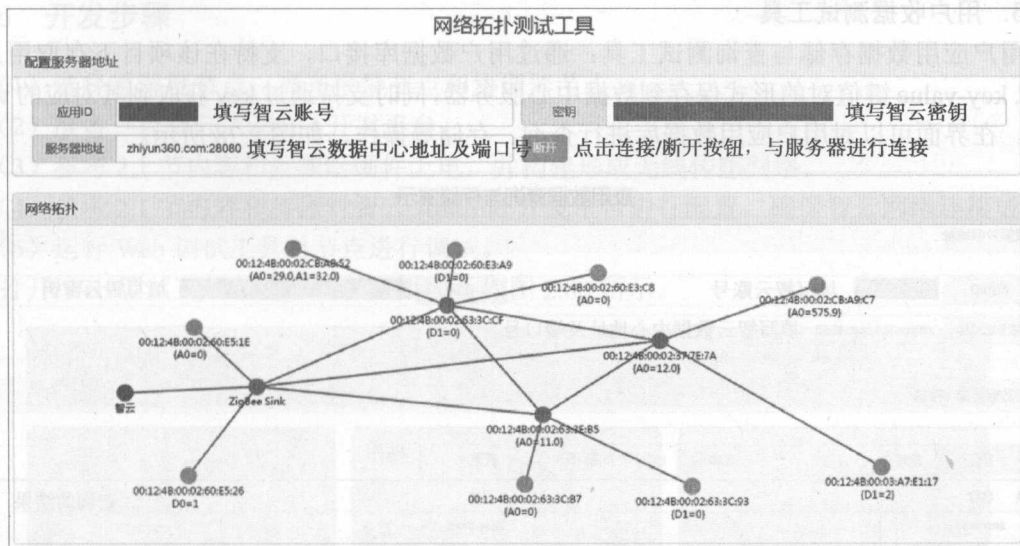


图 2.76 网络拓扑图分析演示

4. 视频监控测试工具

视频监控测试工具：支持对项目中摄像头进行管理，能够实时获取摄像头采集的画面，并支持对摄像头云台进行控制，支持上、下、左、右、水平巡航、垂直巡航等，同时支持截屏操作，如图 2.77 所示。

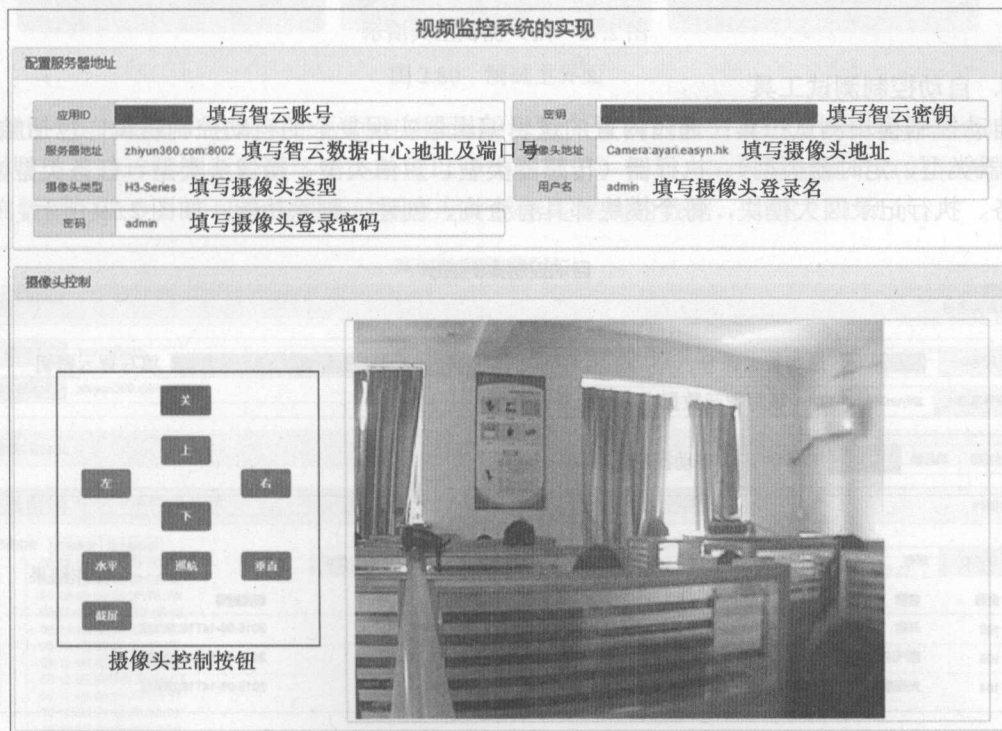


图 2.77 网络拓扑图分析演示



5. 用户收据测试工具

用户应用数据存储与查询测试工具：通过用户数据库接口，支持在该项目下存取用户数据，以 key-value 键值对的形式保存到数据中心服务器，同时支持通过 key 获取到其对应的 value 数值。在界面可以对用户应用数据库进行查询、存储等操作，如图 2.78 所示。

应用数据查询与存储演示

配置服务器地址

应用ID

填写智云账号

密码

填写智云密钥

服务器地址

zhilyun360.com:8080

填写智云数据中心地址及端口号

应用数据查询与存储

操作

获取

名称

名称为空查询所有

查询

操作

ID	名称	值	查询的结果
0	sasdfsad	dfasdfsad	
1	asdfsas	ssfsdsdssds	
2	asdfsasss	ssfsdsdssds	
3	zonesion	中文识别吗？	
4	中文识别吗	1234abcd, ././中文识别吗？	
5	nj-sensor	["00:12:4B:00:02:60:E5:26":{"name":"小区南大门"},"00:12:4B:00:02:60:E5:27":{"name":"小区北大门"}]	
6	nj-user	["BC3323A6":{"name":"用户01","phone":"1864334543","address":"1栋1-304"},"BC3323A7":{"name":"用户02","phone":"13943466606","address":"1栋1-401"},"3A923C0B":{"name":"用户03","phone":"13339873233","address":"2栋1-502"},"DB7EFC07":{"name":"测试卡片","phone":"13224567655","address":"2栋2-304"}]	
7	user	["123":{"name":"test","phone":"1234434"},"456":{"name":"233","phone":"3323"},"83D7E901":{"name":"张三","phone":"13122122132"},"3A923C0B":{"name":"徐志远","phone":"13123346755"},"CB7EFC07":{"name":"李四","phone":"13233434344"},"83D0EC01":{"name":"曾文序","phone":"13233203345"}]	

图 2.78 用户测试工具演示

6. 自动控制测试工具

自动控制模块测试工具：通过内置的逻辑编辑器实现复杂的自动控制逻辑，包括触发器（传感器类型、定时器类型）、执行器（传感器类型、短信类型、摄像头类型、任务类型）、执行任务、执行记录四大模块，每个模块都具有查询、创建、删除功能，如图 2.79 所示。

自动控制案例演示

配置查询信息

应用ID

填写智云账号

密码

填写智云密钥

服务器地址

zhilyun360.com:8001

填写智云数据中心地址及端口号

触发器

执行器

执行任务

执行记录

自动控制选项

任务操作

任务ID	名称	状态	参数	创建时间	操作结果
102	开启	true	{ "aids":108,"ids":242 }	2015-08-14T16:16:31Z	
103	燃气浓度过高，开启报警器	true	{ "aids":109,"ids":243 }	2015-08-14T16:20:02Z	
104	光照强度过高，开启电机	true	{ "aids":110,"ids":244 }	2015-08-14T16:23:04Z	

图 2.79 自动控制工具演示



2.6.5 开发步骤

- (1) 选择温湿度节点、声光报警节点和协调器节点。
 - (2) 准备一台智云 Android 开发平台。
 - (3) 参考 2.1 节内容和步骤给硬件上电，并构建形成无线传感网络。
 - (4) 参考 2.1 节内容和步骤对智云 Android 开发平台进行配置，确保智云网络连接成功。
 - (5) 运行 Web 调试工具对节点进行调试。
- 打开 Web 调试工具，进入调试主页面，如图 2.80 所示。

欢迎进入API测试页面!

首页

实时数据

历史数据

网络拓扑

视频监控

应用数据

自动控制

实时数据

实时数据推送与采集测试工具。通过消息推送接口，能够实时抓取项目上下行所有节点数据包，支持通过命令对节点进行操作，获取节点实时信息、控制节点状态等操作。

历史数据

历数值/图片性历史数据获取测试工具。能够接入数据中心数据库，对项目任意时间段历史数据进行获取，支持数据型数据曲线图展示、JSON数据格式展示，同时支持摄像头抓拍的照片在曲线时间轴展示。

网络拓扑

ZigBee协议模式下网络拓扑图分析工具。能够实时接收并解析ZigBee网络数据包，将接收的网络信息通过拓扑图的形式展示，通过颜色对不同节点类型进行区分，显示节点的IEEE地址。

视频监控

视视频监控测试工具。支持对项目摄像头进行管理，能够实时获取摄像头采集的画面，并支持对摄像头云台进行控制，支持上、下、左、右水平巡航、垂直巡航等，同时支持截屏操作。

用户数据

用户应用数据存储与查询测试工具。通过用户数据库接口，支持在该项目下存取用户数据，以key-value键值对的形式保存到数据中心服务器，同时支持通过key获取到其对应的value数值。

自动控制

自动控制模块测试工具。通过内置的逻辑编辑器实现复杂的自动控制逻辑，包括触发器（传感器类型、定时器类型）、执行器（传感器类型、短信类型、摄像头类型、任务类型）、执行任务、执行记录四大模块，每个模块都具有查询、创建、删除功能。

图 2.80 测试主界面

单击“实时数据”进入实时数据调试页面，先输入正确的智云 ID/KEY 和服务器地址，单击“链接”按钮连接到至云服务器，就可以开始测试了。例如，输入温湿度节点的 MAC 地址和查询数据的命令，即可查询到当前温湿度传感器采集到的数据，如图 2.81 所示。

实时数据测试工具

配置服务器地址

应用ID

1155223953

密码

Xrk6UicNrbo?KY+VYDdLqH4M6RVneF?ShdNlVFYdNlHE

服务器地址

zhiyun360.com:28080

断开

数据推送与接收

地址

00:12:4B:00:02:CB:A8:52

数据

{A0=? ,A1=?}

发送

数据过滤

所有数据

清空全部数据

MAC地址	信息	时间
00:12:4B:00:02:CB:A8:52	{A0=28.0,A1=33.0}	2015/9/16 11:29:23

图 2.81 数据查询 1



也可以输入声光报警器的 MAC 地址和控制报警器的命令，即可对报警器进行实时控制，并查询到报警器当前的状态，如图 2.82 所示。

实时数据测试工具

配置服务器地址

应用ID1155223953

密码Xrk6UicNrbo3KX1tYDDaUq9HAMH6YhuE2Sb4NLKFKdNcLH5

服务器地址zhiyun360.com:28080

断开

数据推送与接收

地址00:12:4B:00:02:63:3C:CF

数据{OD1=1,D1=?}

发送

数据过滤所有数据清空全部数据

MAC地址	信息	时间
00:12:4B:00:02:63:3C:CF	{D1=1}	2015/9/16 11:31:48
00:12:4B:00:02:60:F5:1F		

图 2.82 数据查询 2

单击“用户数据”进入用户数据接口的调试页面，用户数据接口提供存储和读取的操作，可以先选择“存储”。例如，存储数据的 key 为“username”，value 为“刘德华”。存储成功后，再切换至“获取”选项，输入要查询数据的 key，即 username，下面会显示其对应的 value 值，如图 2.83 所示。

用户数据测试工具

配置服务器地址

应用ID11

密码Xrk6UicN

服务器地址zhiyun360.com:8080

用户数据查询与存储

操作获取

名称username

查询

ID	名称	值
1	username	刘德华

图 2.83 数据查询 3

2.6.6 总结与拓展

本任务介绍了云平台的实时推送测试工具、历史数据测试工具、网络拓扑分析工具、视频监控测试工具、用户收据测试工具、自动控制测试工具，可以选择更多的传感器，用 Web 工具进行调试分析。

2.7 任务 11：掌握应用项目上传

2.7.1 学习目标

- 掌握物联网应用项目的开发与发布。



2.7.2 开发环境

硬件：温度传感器 1 个，声光报警传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 2 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

2.7.3 原理学习

智云平台为开发者提供一个应用项目分享的应用网站：<http://www.zhiyun360.com>，通过注册开发者可以轻松发布自己的应用项目。

开发者的应用项目可以展示节点采集的实时在线数据、查询历史数据，并以曲线的方式进行展示；对执行设备，开发者可以编辑控制命令，对设备进行远程控制；同时可以在线查阅视频图像，并且支持远程控制摄像头云台的转动，支持设置自动控制逻辑进行摄像头图片的抓拍并曲线展示，如图 2.84 所示。

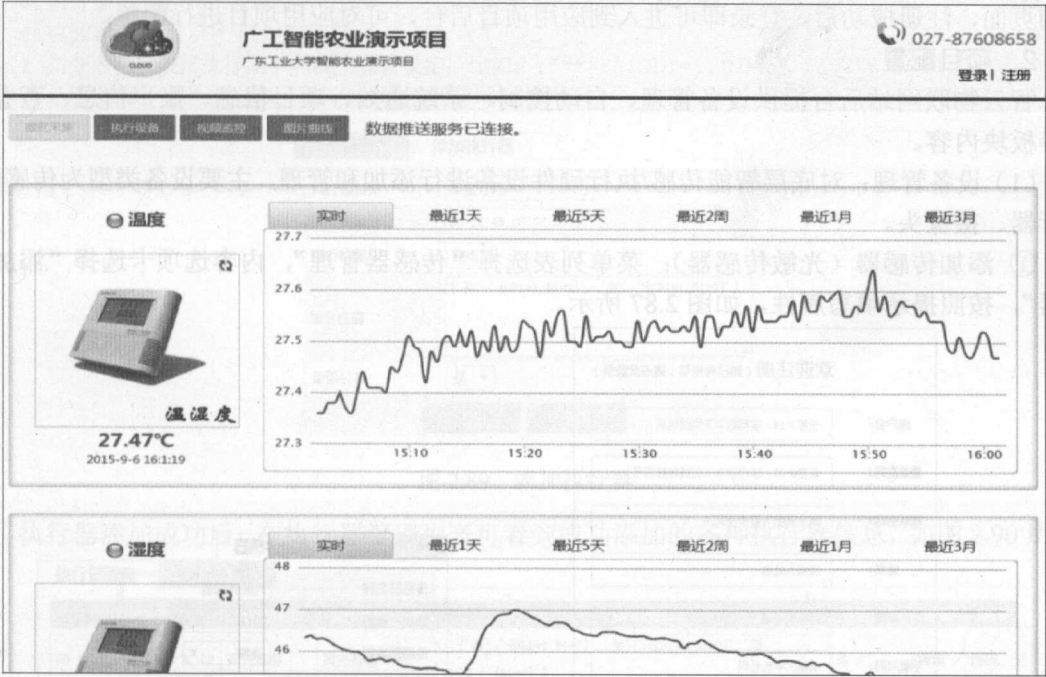


图 2.84 智云应用项目

2.7.4 开发内容

登录智云物联应用网站：<http://www.zhiyun360.com>，如图 2.85 所示。



图 2.85 智云物网站

1. 开发者注册

新开发者需要对应用项目进行注册，在网站右上角单击“注册”按钮，显示如图 2.86 所示的页面，注册成功后，登录即可进入到应用项目后台，可对应用项目进行配置。

2. 项目配置

智云物网站后台提供设备管理、自动控制、系统通知、项目信息、账户信息、查看项目等板块内容。

(1) 设备管理：对底层智能传感/执行硬件设备进行添加和管理，主要设备类型为传感器、执行器、摄像头。

① 添加传感器（光敏传感器）：菜单列表选择“传感器管理”，内容选项卡选择“添加传感器”，按照提示填写属性，如图 2.87 所示。

欢迎注册 (如有帐号, 请点此登录)

用户名:

长度3-14, 字母数字下划线组成

登录密码:

长度6-20, 字母数字下划线特殊符号

确认密码:

请从新输入登录密码

姓名:

请输入姓名

邮箱:

请输入邮箱

手机号码:

请输入手机号码

注册

图 2.86 开发者注册

传感器管理

添加传感器

传感器名称	光照传感器
数据流通道	00:12:4B:00:02:63:3C:4F_A0
传感器类型	光照
曲线形状	平滑
是否公开	是

添加

返回

图 2.87 添加传感器

- # 传感器名称: 开发者为设备自定义的名称
- # 数据流通道: "IEEE 地址_通道名" (IEEE 读取方法见附录 A.4), 比如: 00:12:4B:00:02:63:3C:4F_A0
- # 传感器类型: 从下拉列表选择
- # 曲线形状: 模拟量类传感器可选择“平滑”, 电平类传感器可选择“阶梯”
- # 是否公开: 是否将该传感器信息展示到前端项目网页



传感器添加成功后，在传感器管理列表可看到成功添加的各种传感器信息，如图 2.88 所示。

传感器管理		添加传感器					
通道	传感器名称	传感器类型	单位	曲线类型	是否公开	编辑	删除
00:12:4B:00:02:CB:A8:52_A0	温度传感器	温度	℃	平滑	是	编辑	删除
00:12:4B:00:02:CB:A8:52_A1	湿度传感器	湿度	%	平滑	是	编辑	删除
00:12:4B:00:02:CB:A9:C7_A0	光照传感器	光照	LF	平滑	是	编辑	删除
00:12:4B:00:02:63:3E:B5_A0	空气质量传感器	空气质量	ppm	平滑	是	编辑	删除
00:12:4B:00:02:60:FB:67_A0	湖南演示燃气	可燃气体		平滑	否	编辑	删除
00:12:4B:00:02:63:3A:FC_A0	湖南演示温度	温度	℃	平滑	否	编辑	删除
00:12:4B:00:02:63:3A:FC_A1	湖南演示湿度	湿度	%	平滑	否	编辑	删除

图 2.88 传感器信息

② 添加执行器（继电器传感器）：菜单列表选择“执行器管理”，内容选项卡选择“添加执行器”，按照提示填写属性，如图 2.89 所示。

- # 执行器名称：开发者为设备自定义的名称
- # 执行器地址：IEEE 地址，比如：00:12:4B:00:02:63:3C:4F
- # 执行器类型：从下拉列表选择
- # 指令内容：根据执行器节点程序逻辑设定，比如：{'开': '{OD1=1, D1=?}', '关': '{CD1=1, D1=?}'}
- # 是否公开：是否将该执行器信息展示到前端项目网页

执行器管理

添加执行器

执行器名称

卧室灯光

执行器地址

00:12:4B:00:02:63:3C:

执行类型

继电器

指令内容

{'开': '{OD1=1, D1=?}', '关': '{CD1=1, D1=?}'}

是否公开

是

添加

返回

图 2.89 添加执行器

执行器添加成功后，在执行器管理列表可看到成功添加的各种执行器信息，如图 2.90 所示。

执行器管理		添加执行器					
执行器地址	执行器名称	执行类型	单位	指令内容	是否公开	编辑	删除
00:12:4B:00:02:63:3C:CF	声光报警	声光报警		{'开': '{OD1=1, D1=?}', '关': '{CD1=1, D1=?}', '查询': '{D1=?}'}	是	编辑	删除
00:12:4B:00:02:60:B5:1E	步进电机	步进电机		{'正转': '{OD1=3, D1=?}', '反转': '{CD1=2, OD1=1, D1=?}', '停止': '{CD1=1, D1=?}', '查询': '{D1=?}'}	是	编辑	删除
00:12:4B:00:02:60:E3:A9	风扇	风扇		{'开': '{OD1=1, D1=?}', '关': '{CD1=1, D1=?}', '查询': '{D1=?}'}	是	编辑	删除
00:12:4B:00:02:60:B5:26	RFID	低频RFID		{'开': '{ODO=1, DO=?}', '关': '{CDO=1, DO=?}', '查询': '{DO=?}'}	是	编辑	删除
00:12:4B:00:02:63:3C:4F	卧室灯光	继电器		{'开': '{OD1=1, D1=?}', '关': '{CD1=1, D1=?}'}	是	编辑	删除

图 2.90 执行器信息



③ 添加摄像头：菜单列表选择“摄像头管理”，内容选项卡选择“添加摄像头”，按照提示填写属性，如图 2.91 所示。

- # 摄像头名称：开发者为设备自定义的名称
- # 摄像头 IP：从摄像头底部的条码标签可获取
- # 摄像头用户名：根据摄像头的配置设定
- # 摄像头密码：根据摄像头的配置设定
- # 是否公开：是否将该摄像头信息展示到前端项目网页

摄像头管理

添加摄像头

摄像头名称

武汉办公室

摄像头ip

http://p2.ipcam.hk

摄像头类型

F-Series

摄像头用户名

admin

摄像头密码

是否公开

是

添加

返回

图 2.91 添加摄像头

摄像头添加成功后，在摄像头管理列表可看到成功添加的各种摄像头信息，如图 2.92 所示。

摄像头管理

添加摄像头

摄像头名称	摄像头类型	摄像头IP	是否公开	摄像头用户名	摄像头密码	编辑	删除
会议室摄像头	F-Series	217022.easyn.hk	是	admin	admin	编辑	删除
培训摄像头	F3-Series	069208.ipcam.hk	否	admin		编辑	删除

图 2.92 摄像头信息

至此项目设备配置完成。

(2) 自动控制：本版块内容较为复杂，具体先阅读《智云 API 编程手册》进行了解，界面信息如图 2.93 所示。

云物联 ZCloud

设备管理

自动控制

系统通知

项目信息

账户信息

查看项目

用户组

普通用户

菜单列表

触发器管理

控制器管理

执行任务管理

任务执行记录

触发器管理

添加触发器

ID	名称	参数	类型	创建时间	操作
95	timer_every_10_minute	{"once":true,"mac":"00:12:48:00:01:43:17:9F","ch":"A0","value":"20","op":"="}	sensor	2015-06-11T17:26:15Z	删除
97	per30_minute	{"week":"*","hour":"*","month":"*","second":"0","year":"*","day":"*","minute":"30"}	timer	2015-06-17T09:41:52Z	删除

总2条，当前第0-2条 首页 | 上一页 | 下一页 | 尾页

图 2.93 自动控制

(3) 系统通知：本版块是由网站系统管理发布的一些通知信息。

(4) 项目信息：本版块是用于描述开发者应用项目信息，项目信息是对应用项目名称、副标题、介绍等的描述，上传图像是提交开发者应用项目的 Logo 图标。智云 ID/KEY 要求



填写与项目所在网关一致的正确授权的智云 ID/KEY。地理位置可在地图页面标记自己的位置：输入所在城市的中文名称进行搜索，然后在地图小范围确定地点，如图 2.94 和图 2.95 所示。

项目信息

上传图像

项目名称	武汉理工大学智能家居
项目副标题	实现对家居环境的远程采集，包括温湿度、光照度、空气质量等
用户主页网址	
项目介绍	<p>本项目主要是实现一个远程观测家居环境的系统，能够检测温度、湿度、空气质量等参数。在前端的web页面可以看到几个传感器参数的历史数值，通过曲线图的形式表现，另外也会实时推送最新的数据到web端。</p> <p>整个项目基于智云物联ZCloud云平台架构开发。</p> <p>开发者：lusi</p>
地理位置	经度：114.345916 纬度：30.519038
智云ID	
智云KEY	
数据中心地址	zhiyun360.com
允许添加的传感器总数	30
允许添加的摄像头总数	30
允许添加的执行器总数	30
<div>编辑项目信息</div>	

图 2.94 项目信息 1

编辑项目信息

项目名称	武汉理工大学智能家居
项目副标题	实现对家居环境的远程采集，包括温湿度、光照度、空气质量等
用户主页网址	<div>网址中带有http://</div>
项目介绍	<div><div><div>B I U S X</div><div>加粗 斜体 下划线 删除 格式刷</div></div><div><p>本项目主要是实现一个远程观测家居环境的系统，能够检测温度、湿度、空气质量等参数。在前端的web页面可以看到几个传感器参数的历史数值，通过曲线图的形式表现，另外也会实时推送最新的数据到web端。</p><p>整个项目基于智云物联ZCloud云平台架构开发。</p><p>开发者：lusi</p></div></div>
数据中心地址	<div>zhiyun360.com</div> <div>示例：zhiyun360.com</div>
智云账号	此处填写与智云网关一致的正确智云ID/KEY
智云密钥	
<div><div>搜索 输入城市名(汉字)</div><div>柏景街道</div><div>经度：114.345916 纬度：30.519038</div></div>	

图 2.95 项目信息 2

(5) 账户信息：“账户信息”栏目用于用户信息的填写，将在用户项目的首页底部展示，如图 2.96 所示。



基本信息	
用户名	smarthome
姓名	卢工
用户地址	
电子邮箱	lusi@zrnesion.com.cn
手机号码	18164011850
编辑个人信息	

图 2.96 账户信息

(6) 查看项目：单击“查看项目”栏目即可进入到开发者所在项目的首页。

3. 项目发布

开发者的项目配置好了，即完成了项目的发布，在开发者项目后台可设置各种设备的公开权限，普通开发者在项目页面无法浏览禁止公开的设备。

项目展示页示例如下所示 (<http://www.zhiyun360.com/Home/Sensor?ID=46>)。

(1) 查看传感器数据：数据采集选项卡，左边栏显示传感器图片、名称、实时接收到的数值、在线状态（在线则传感器名称的指示图标变蓝色，不在线为灰色），右边栏显示传感器一段时间内的数据曲线，可单击标签选择：实时（最近1小时）、最近1天、最近5天、最近2周、最近1月、最近3月的数据，如图 2.97 所示。



图 2.97 传感器信息



(2) 实时控制执行器：执行设备选项卡，左边栏显示执行器图片、名称、在线状态（在线则传感器名称的指示图标变蓝色，不在线为灰色），右边栏显示该执行器可进行的操作，单击对应的按钮，可远程对设备进行控制，同时在“反馈信息”窗口可查看控制命令及反馈的消息结果，如图 2.98 所示。

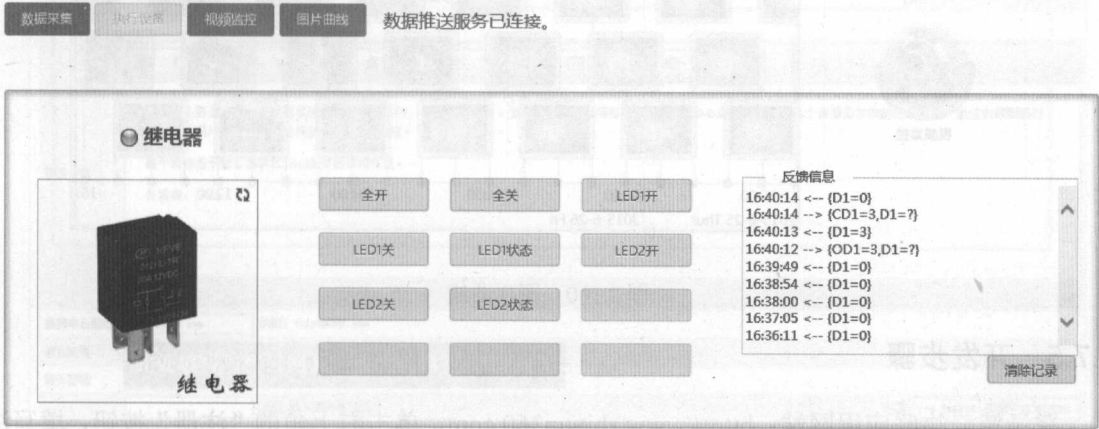


图 2.98 设备控制信息

(3) 视频监控：视频监控选项卡，左边栏显示摄像头图片、名称、在线状态（在线则传感器名称的指示图标变蓝色，不在线为灰色）、控制按钮，右边栏显示摄像头采集的图像画面，单击对应的按钮，可远程对摄像头进行开关及云台转动操作，如图 2.99 所示。

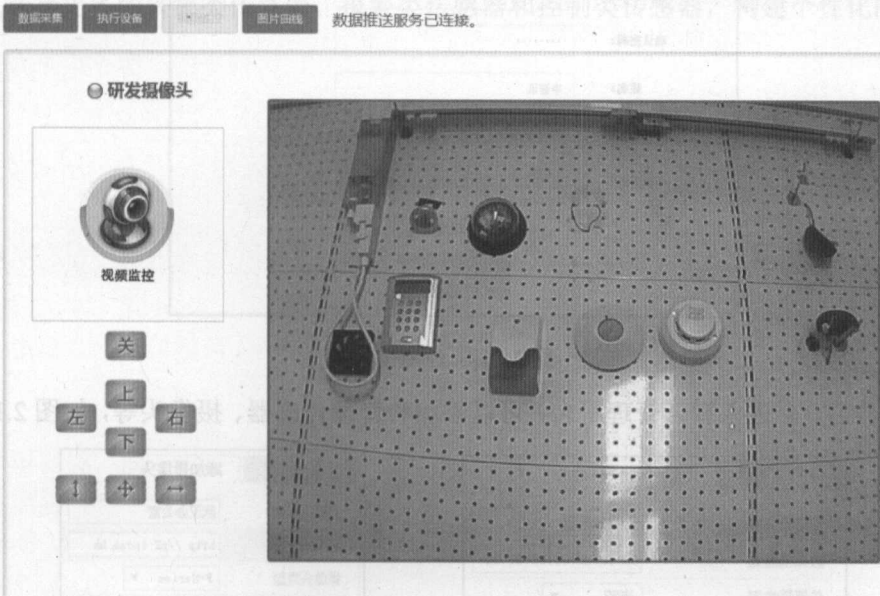


图 2.99 视频监控

(4) 图片曲线：图片曲线选项卡，左边栏显示摄像头图片、名称，右边栏显示摄像头定时抓拍的图片（此功能需要在后台自动控制板块添加策略），如图 2.100 所示。

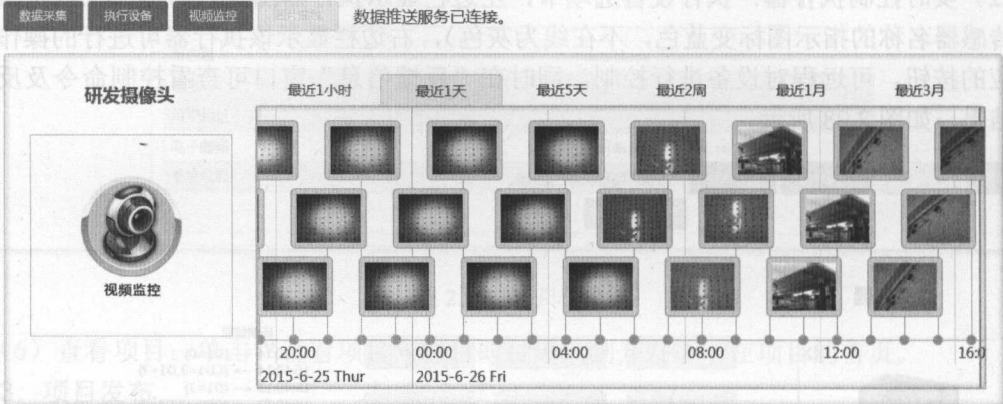


图 2.100 图片曲线

2.7.5 开发步骤

登录智云物联应用网站：<http://www.zhiyun360.com>，单击右上角的“注册”按钮，填写注册信息，如图 2.101 所示。

图 2.101 注册信息

完成注册后，会进入智云管理平台，可以添加和管理传感器、摄像头等，如图 2.102 所示。

图 2.102 添加设备信息

图 2.103 所示。

编辑项目信息

项目名称
武汉理工大学智能家居

项目副标题
实现对家居环境的远程采集，包括温湿度、光照度、空气质量等

用户主页网址
网址中带有http://

B I U S x^{*} | 列表图标 | 全屏图标 | 打印图标 | 分享图标 | 撤销图标 | 重做图标 | 格式刷图标

本项目主要是实现一个远程观测家居环境的系统，能够检测温度、湿度、空气质量等参数。在前端的Web页面可以看到几个传感器参数的历史数值，通过曲线图的形式表现，另外也会实时推送最新的数据到Web端。

整个项目基于智云物联ZCloud云平台架构开发。

开发者：luzi

数据中心地址
zhiyun360.com 示例：zhiiyun360.com

智云账号
此处填写与智云网关一致的正确智云ID/KEY

智云密钥

搜索 输入城市名(汉字)

柏泉街道

外环线

111

220

经纬度 114.345916 纬度 30.619038

图 2.103 添加项目信息

2.7.6 总结与拓展

可以选择更多的采集类传感器、报警类传感器和控制类传感器，构建个性化的应用项目。

智云物联综合应用开发

本章通过 7 个案例，完成智云物联平台独立模块的硬件资源、使用说明及各种运行模式，从环境搭建到硬件驱动，再到移动端开发和 Web 端开发，一步步引导读者实现各种智能控制功能，学习完后，开发者将初步具有物联网和云平台设计能力。

3.1 任务 12：远程温湿度计系统开发（案例 1）

3.1.1 学习目标

- 掌握传感器数据的定时上报采集；
- 掌握实时数据编程接口的使用；
- 掌握通信协议 sensor_update()函数的运用；
- 学会远程温湿度系统项目开发与调试。

3.1.2 开发环境

硬件：温度传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

3.1.3 原理学习

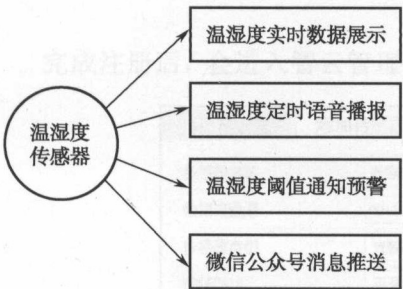


图 3.1 远程温湿度计功能模块

1. 系统设计目标

温湿度作为典型的物联网传感器，能够定时采集环境温度湿度数值并主动上传。通过对空气温湿度传感器的采集监控，实现智能数字温湿度计的设计，能够实时将温湿度信息推送到 Android 移动客户端，实现随时随地远程获取家居温湿度等环境信息，系统设计功能及目标如图 3.1 所示。

2. 业务流程分析

远程数字温度计从传输过程分为三部分：传感节点、网关、客户端（Android 或 Web），



通信流程如图 3.2 所示，具体通信描述如下。

- (1) 传感器节点通过 ZigBee 网络与网关的协调器进行组网，网关的协调器通过串口与网关进行数据通信；
- (2) 底层节点的数据通过 ZigBee 网络将数据传送给协调器，协调器通过串口将数据转发给网关服务，通过实时数据推送服务将数据推送给所有连接网关的客户端；
- (3) Android 应用通过调用 ZCloud SDK API 的实时数据连接接口实现实时数据采集的功能。

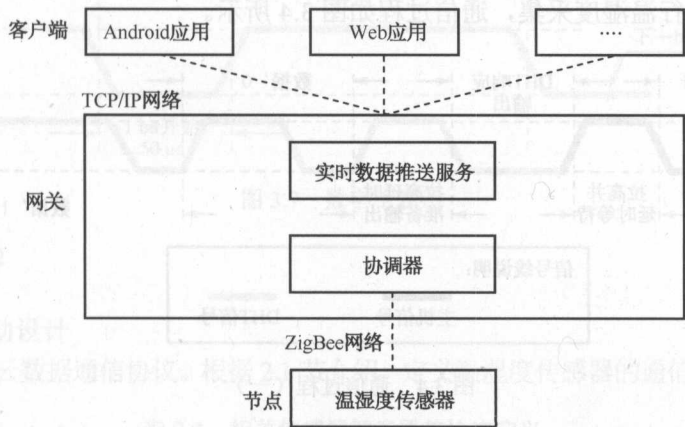


图 3.2 远程温湿度计程序流程

3. 硬件原理

本任务中采用的传感器为 DHT11 数字温湿度传感器，DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性与卓越的长期稳定性。传感器包括一个电阻式感湿元件和一个 NTC 测温元件，并与一个高性能 8 位单片机相连接。因此该产品具有品质卓越、超快响应、抗干扰能力强、性价比高等优点。每个 DHT11 传感器都在极为精确的湿度校验室中进行校准。校准系数以程序的形式储存在 OTP 内存中，传感器内部在检测信号的处理过程中要调用这些校准系数。单线制串行接口，使系统集成变得简易快捷。超小的体积、极低的功耗，信号传输距离可达 20 m 以上，使其成为各类应用甚至最为苛刻的应用场合的最佳选则。

通过 CC2530 IO 口模拟 DHT11 的读取时序，读取 DHT11 的温湿度数据，如图 3.3 所示。

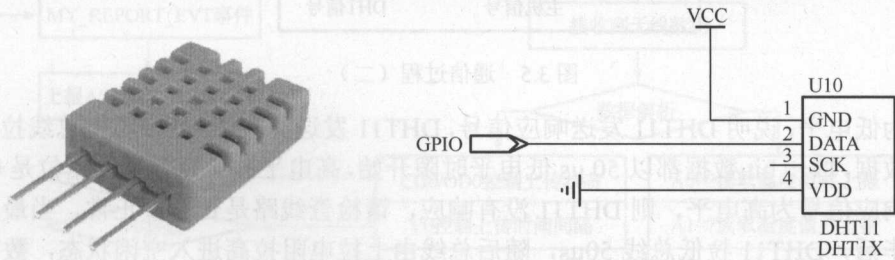


图 3.3 温湿度传感器硬件原理

DHT11 的获取温湿度值的原理：DHT11 的串行接口 DATA 用于微处理器与 DHT11 之间的通讯和同步，采用单总线数据格式，一次通讯时间 4ms 左右，数据分小数部分和整数部分，



具体格式在下面说明，当前小数部分用于以后扩展，现读出为零。操作流程如下。

一次完整的数据传输为 40 bit，数据格式：8 bit 湿度整数数据+8 bit 湿度小数数据+8 bit 温度整数数据+8 bit 温度小数数据+8 bit 校验和数据传送正确时校验和数据等于“8 bit 湿度整数数据+8 bit 湿度小数数据+8 bit 温度整数数据+8 bit 温度小数数据”所得结果的末 8 位。CC2530 发送一次开始信号后，DHT11 从低功耗模式转换到高速模式，等待主机开始信号结束后，DHT11 发送响应信号，送出 40 bit 的数据，并触发一次信号采集，可选择读取部分数据。从模式下，DHT11 接收到开始信号触发一次温湿度采集，如果没有接收到主机发送开始信号，DHT11 不会主动进行温湿度采集，通信过程如图 3.4 所示。

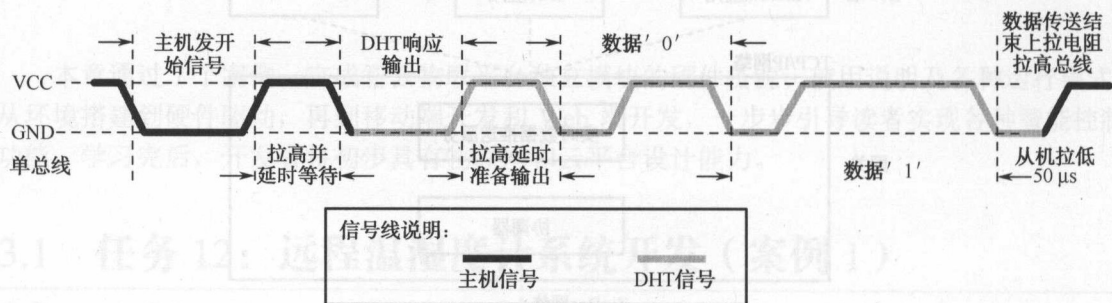


图 3.4 通信过程（一）

总线空闲状态为高电平，主机把总线拉低等待 DHT11 响应，主机把总线拉低必须大于 18ms，保证 DHT11 能检测到起始信号。DHT11 接收到主机的开始信号后，等待主机开始信号结束，然后发送 80 μs 低电平响应信号。主机发送开始信号结束后，延时等待 20~40 μs 后，读取 DHT11 的响应信号，主机发送开始信号后，可以切换到输入模式，或者输出高电平均可，总线由上拉电阻拉高，如图 3.5 所示。

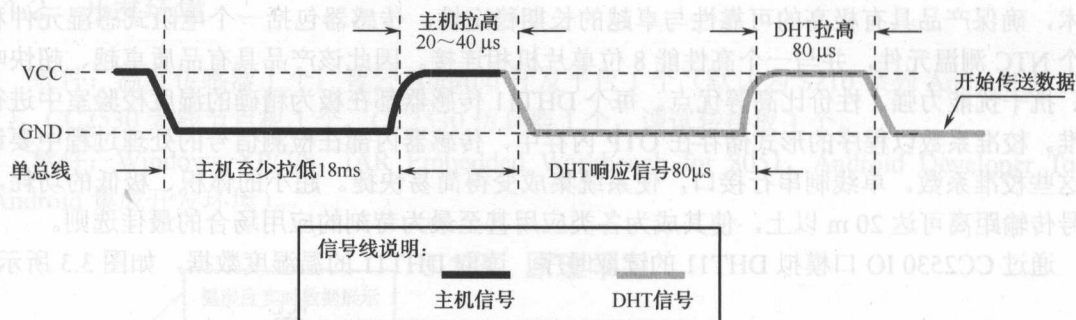


图 3.5 通信过程（二）

总线为低电平，说明 DHT11 发送响应信号，DHT11 发送响应信号后，再把总线拉高 80 μs，准备发送数据，每一 bit 数据都以 50 μs 低电平时隙开始，高电平的长短定了数据位是 0 还是 1。如果读取响应信号为高电平，则 DHT11 没有响应，请检查线路是否连接正常。当最后一位数据传送完毕后，DHT11 拉低总线 50μs，随后总线由上拉电阻拉高进入空闲状态，数字 0 信号表示方法如图 3.6 所示。

数字 1 信号表示方法如图 3.7 所示。

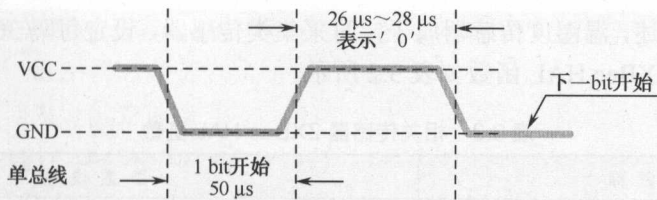


图 3.6 数字 0 表示

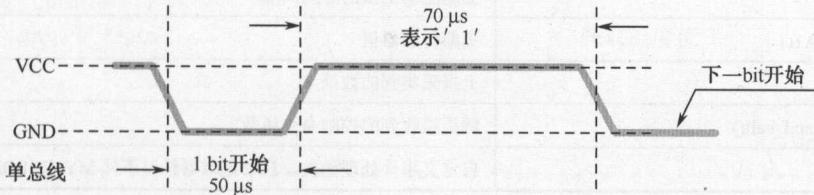


图 3.7 数字 1 表示

3.1.4 开发内容

1. 硬件层驱动设计

(1) ZXBee 智云数据通信协议。根据 2.1 节介绍，定义温湿度传感器的通信协议如表 3.1 所示。

表 3.1 相关传感器智云通信协议定义

传 感 器	属 性	参 数	权限	说 明
温湿度	温度值	A0	R	温度值，浮点型：0.1 精度
	湿度值	A1	R	湿度值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示温度上传状态、Bit1 表示湿度上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔

(2) 传感器驱动程序开发。温湿度传感器程序逻辑如图 3.8 所示。

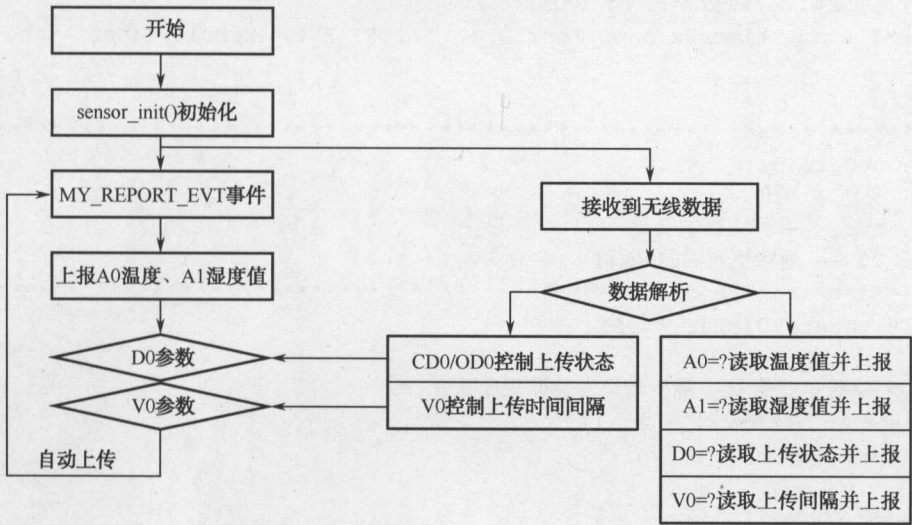


图 3.8 远程温湿度计程序逻辑



根据 2.1 章节所述，温湿度传感器属于定时采集类传感器，设定每隔 30 s 主动上报传感器数值。相关传感器 ZXBee HAL 函数如表 3.2 所示。

表 3.2 相关传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	初始化传感器最基本的是配置选择寄存器和方向寄存器
updateV0()	更新主动上报的时间间隔
updateA0()/updateA1()	更新传感器值
sensor_update()	上报采集到的数据
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	自定义事件处理函数，启动定时器触发事件 MY_REPORT_EVT

部分程序代码如下。

```

/*****全局变量*****/
static uint8 D0 = 3;           //默认打开主动上报功能
static float A0 = 0.0;        //A0 存储温度值
static float A1 = 0.0;        //A1 存储湿度值
static uint16 V0 = 30;        //V0 设置为上报时间间隔，默认为 30 s
static uint16 myReportInterval = 30; //上报时间间隔，单位为 s

/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    dht11_io_init();

    //启动定时器，触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10) * 1000));
}

/*****
*名称: updateV0()
*功能: 更新 V0 的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的 V0 值
*****/
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}
/*****
```



```

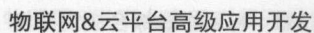
*名称: updateA0()
*功能: 更新 A0 的值
*参数: 无
*返回: A0 -- 返回更新后的 A0 值
*****/
float updateA0(void)
{
    dht11_update();
    A0 = dht11_temp();                //获取温度值

    return A0;
}
/*****
*名称: updateA1()
*功能: 更新 A1 的值
*参数: 无
*返回: A1 -- 返回更新后的 A1 值
*****/
float updateA1(void)
{
    dht11_update();
    A1 = dht11_humidity();           //获取湿度值

    return A1;
}
/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){        //若温度上报允许, 则 pData 的数据包中添加温度数据
        updateA0();
        len = sprintf((char*)p, "A0=%.1f", A0);
        p += len;
        *p++ = ',';
    }
    if ((D0 & 0x02) == 0x02){        //若湿度上报允许, 则 pData 的数据包中添加湿度数据
        updateA1();
        len = sprintf((char*)p, "A1=%.1f", A1);
        p += len;
        *p++ = ',';
    }
}

```



//更新温度数值

//更新湿度数值



```
        ret = sprintf(pout, "V0=%u", V0);
    }else{
        updateV0(pval);
    }
}

return ret;
}

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器，触发事件: MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
                                                                    *1000));
    }
}
```

2. 移动端应用设计

(1) 工程框架介绍。工程框架如表 3.3 所示。

表 3.3 远程数字温湿度计工程框架介绍

包名（类名）	说 明
com.zonesion.temphumidemo 应用包	
temphumiActivity.java	传感器处理主程序
AboutActivity.java	智云物联介绍页面
activity_temphumi.xml	主界面布局文件

(2) 程序业务流程分析。根据 2.1 节智云 Android 应用编程接口定义，远程温湿度的应用设计主要采用实时数据 API 接口，程序流程如图 3.9 所示。

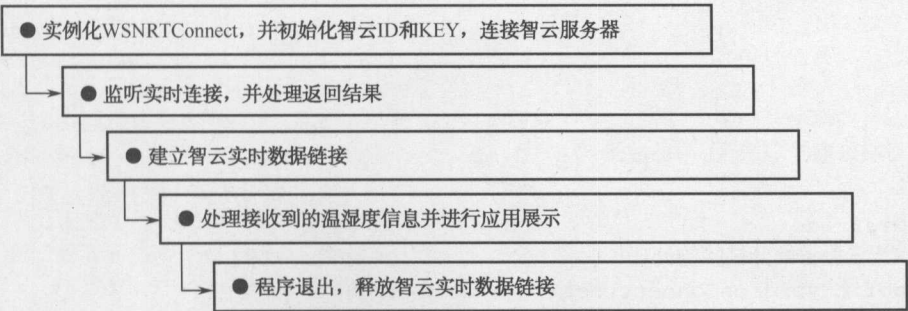


图 3.9 Android 应用程序逻辑



(3) 程序代码剖析。

① 例化 WSNRTConnect, 并初始化智云 ID 和 KEY, 连接智云服务器 (局域网内使用: ID/KEY 可任意填写, 注意每个 ID 不能相同, 服务地址为智云 Android 开发平台的 IP 地址。若链接到互联网访问, 需要填写正确的 ID/KEY/服务地址)。

```
private WSNRTConnect wRTConnect;           //创建 WSNRTConnect 实例
private String mMac = "00:12:4B:00:02:CB:A8:52"; //温湿度传感器节点 IEEE 地址
private String ID = "12345678";           //用户账号
private String KEY = "12345678";          //用户密钥

wRTConnect = new WSNRTConnect(ID, KEY);    //创建 wRTConnect 链接, 并初
始化
wRTConnect.setServerAddr("zhiyun360.com:28081"); //设置智云服务地址
```

② 实监听实时连接, 并处理数据。

```
wRTConnect.setRTConnectListener(new WSNRTConnectListener() { //设置监听
    @Override
    //消息到达时会自动调用该方法
    public void onMessageArrive(String mac, byte[] data) {
        if (mMac.equalsIgnoreCase(mac)) { //判断 MAC 地址
            //解析数据
            if (data[0] == '{' && data[data.length - 1] == '}') {
                Toast.makeText(temphumiActivity.this, "获取到传感器上传的数据",
                    Toast.LENGTH_SHORT).show();
                String sData = new String(data, 1, data.length - 2);
                String[] pDatas = sData.split(",");
                for (String pData : pDatas) {
                    String[] tagVal = pData.split("=");
                    if (tagVal.length == 2) {
                        if (tagVal[0].equals("A0")) { //判断参数
                            float v = Float.parseFloat(tagVal[1]);
                            temperatureTextView.setText("温度值: "+fnum.format(v)+
                                "℃");
                        }
                        if (tagVal[0].equals("A1")) { //判断参数
                            float v = Float.parseFloat(tagVal[1]);
                            humidityTextView.setText("湿度值"+fnum.format(v) + "%");
                        }
                    }
                }
            }
        }
    }

    @Override
    //智云服务断开链接的处理
    public void onConnectLost(Throwable arg0) {
        Toast.makeText(temphumiActivity.this, "断开连接", Toast.LENGTH_LONG).
            show();
    }
    @Override
```



```
//智云服务链接建立成功处理
public void onConnect() {
    //TODO Auto-generated method stub
    Toast.makeText(temphumiActivity.this, "连接网关成功",
        Toast.LENGTH_SHORT).show();

    if (mMac.length() > 0) {
        wRTConnect.sendMessage(mMac, "{A0=?,A1=?}".getBytes()); //查询当前传感器数值
        Toast.makeText(temphumiActivity.this, "{A0=?,A1=?}",
            Toast.LENGTH_LONG).show();
    }
}
});
```

③ 建立智云实时数据链接。

```
wRTConnect.connect(); //与智云服务建立链接
```

④ 单击“数据采集”按键更新温湿度传感器数据。

//功能：查询当前温湿度传感器数值

```
public void dataCollection(View v) {
    wRTConnect.sendMessage(mMac, "{A0=?,A1=?}".getBytes());
    Toast.makeText(temphumiActivity.this, "{A0=?,A1=?}",
        Toast.LENGTH_LONG).show();
}
```

⑤ 修改应用程序图标：在 AndroidManifest.xml 文件中修改属性 application 的 icon 属性。

```
Android:icon="@drawable/ic_launcher"
```

⑥ 程序退出，释放智云实时数据链接。

```
@Override
public void onDestroy() {
    wRTConnect.disconnect(); //断开智云链接
    super.onDestroy();
}
```

3. Web 端应用设计

根据 Web 应用编程接口定义，远程温湿度的应用设计主要采用实时数据 API 接口，JS 部分控制采集代码如下。

```
<script type="text/javascript">
$(function() {
    var myZCloudID = "12345678"; //序列号
    var myZCloudKey = "12345678"; //密钥
    var mySensorMac = "00:12:4B:00:02:CB:A8:52"; //传感器的 MAC 地址
    var rtc = new WSNRTConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
    rtc.setServerAddr("zhiyun360.com:28080");
    rtc.connect();
    rtc.onConnect = function() { //连接成功回调函数
        rtc.sendMessage(mySensorMac, "{A0=?,A1=?}"); //向传感器发送数据
        $("#ConnectState").text("数据服务连接成功!");
    };

    rtc.onConnectLost = function() { //数据服务掉线回调函数
```




```

$("#ConnectState").text("数据服务掉线!");
};

rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
    if (mac == mySensorMac) { //判断传感器 MAC 地址
        if (dat[0] == '{' && dat[dat.length - 1] == '}') { //判断字符串首尾是否为{}
            dat = dat.substr(1, dat.length - 2); //截取{}内的字符串
            var its = dat.split(','); //以','来分割字符串
            for (var x in its) {
                var t = its[x].split('='); //以'='来分割字符串
                if (t.length != 2) continue;
                if (t[0] == "A0") { //判断参数 A0
                    var tem = parseInt(t[1]);
                    $("#currentTem").text(tem + "°C");
                    //更新图表
                    getDialTem(parseFloat(tem));
                }
                if (t[0] == "A1") { //判断参数 A0
                    var hum = parseInt(t[1]);
                    $("#currentHum").text(hum + "%");
                    //更新图表
                    getDialHum(parseFloat(hum));
                }
            }
        }
    }
};

$("#searchTem").click(function() { //查询温度
    rtc.sendMessage(mySensorMac, "{A0=?}"); //向传感器发送数据
});
$("#searchHum").click(function() { //查询湿度
    rtc.sendMessage(mySensorMac, "{A1=?}"); //向传感器发送数据
});
});
</script>

```

3.1.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个温湿度传感器无线节点, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到 “C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples” 文件夹下。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。



2. Android 应用程序开发

- (1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。
- (2) 编译 TempHumiDemo 工程，并安装应用程序到 Android 开发平台或 Android 终端内。
- (3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入“远程温湿度计”模块界面，在主界面弹出“连接网关成功”消息后表示链接到智云服务中心，在中间的温度图标下方会实时显示当前的温度值，如图 3.10 所示。



图 3.10 Android 端远程数字温湿度计展示

3. Web 应用程序开发

- (1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。
- (2) 设置计算机接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（支持 HTML5 的 IE10 浏览器）打开本任务的 Web 工程“TempHumiDemo-Web\TempHumiDemo.html”，进入“远程温湿度计”模块界面，在主界面右上角显示“数据服务连接成功!”消息后即表示链接到智云服务中心，在界面可以看到温湿度的显示，如图 3.11 所示。

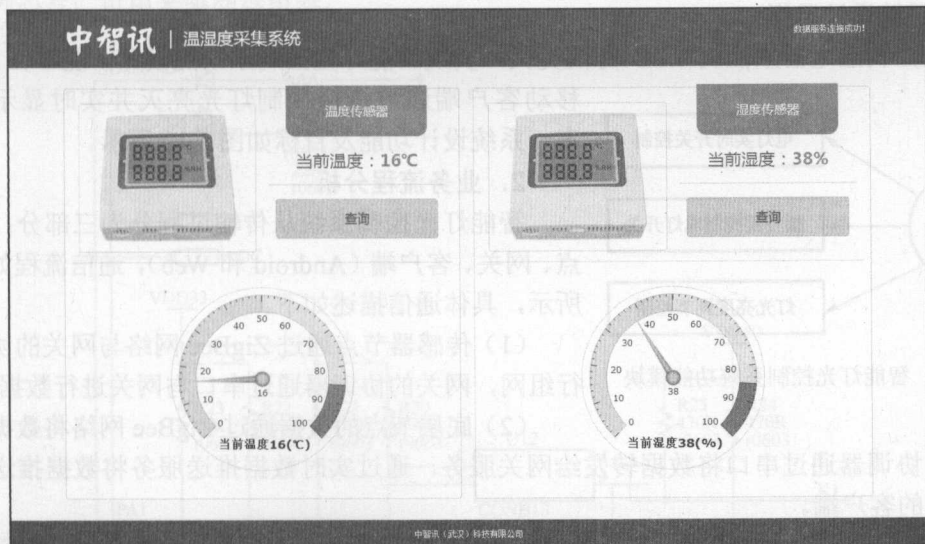


图 3.11 Web 端远程数字温湿度计展示



3.1.6 总结与拓展

实现了温湿度实时数据展示功能，实时获取温湿度传感器的数据显示当前温湿度值，另外实现了 Android 应用程序修改图标的功能；开发者可以自行实现温湿度定时语音播报、温湿度阈值通知预警及微信公众号消息推送等功能。

3.2 任务 13：智能灯光控制系统开发（案例 2）

3.2.1 学习目标

- 掌握控制类传感器的实时控制；
- 掌握实时数据编程接口的使用；
- 掌握通信协议 sensor_control()函数的运用；
- 掌握 AndroidApplication 类的使用；
- 学会智能灯管系统项目开发与调试。

3.2.2 开发环境

硬件：继电器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

3.2.3 原理学习

1. 系统设计目标

通过控制继电器来控制 LED 灯的亮灭，实现智能灯光控制系统的设计，能够在 Android 移动客户端通过按钮控制灯光亮灭并实时显示当前状态，系统设计功能及目标如图 3.12 所示。

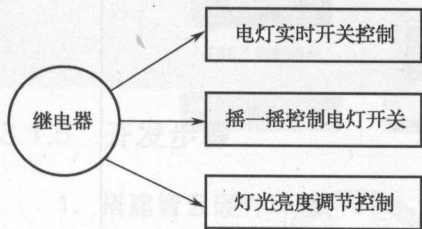


图 3.12 智能灯光控制系统功能模块

2. 业务流程分析

智能灯光控制系统从传输过程分为三部分：传感节点、网关、客户端（Android 和 Web），通信流程如图 3.13 所示，具体通信描述如下。

（1）传感器节点通过 ZigBee 网络与网关的协调器进行组网，网关的协调器通过串口与网关进行数据通信。

（2）底层节点的数据通过 ZigBee 网络将数据传送给协调器，协调器通过串口将数据转发给网关服务，通过实时数据推送服务将数据推送给所有连接网关的客户端。

（3）Android 应用通过调用 ZCloud SDK API 的实时数据连接接口实现实时数据采集的功能，详细实现参考下一节开发内容代码实现部分。

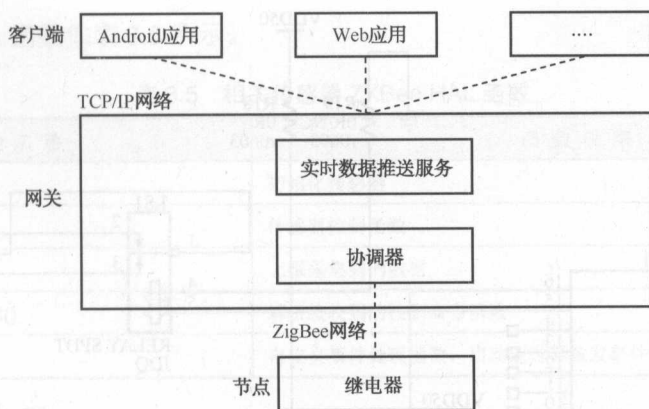


图 3.13 智能灯光控制系统程序流程

3. 硬件原理

本任务使用的继电器模块是电磁继电器，电磁继电器一般由铁芯、线圈、衔铁、触点簧片等组成的。只要在线圈两端加上一定的电压，线圈中就会流过一定的电流，从而产生电磁效应，衔铁就会在电磁力吸引的作用下克服返回弹簧的拉力吸向铁芯，从而带动衔铁的动触点与静触点（常开触点）吸合。当线圈断电后，电磁的吸力也随之消失，衔铁就会在弹簧的反作用力返回原来的位置，使动触点与原来的静触点（常闭触点）释放。这样吸合、释放，从而达到了在电路中的导通、切断的目的。

“常开、常闭”触点：继电器线圈未起电时处于断开状态的静触点，称为“常开触点”；处于接通状态的静触点称为“常闭触点”。

通过 CC2530 的 IO 口输出高低电平实现继电器闭合/断开的控制，继电器模块与 CC2530 部分接口电路如图 3.14 和图 3.15 所示，其中 ADC、GPIO 分别对应 CC2530 的 P0_1、P0_5 两个 IO 口。U12 和 U13 是光耦隔离芯片 TLP281-1，U11 是 ULN2003A 高压大电流达林顿晶体管阵列芯片，可用于驱动继电器。

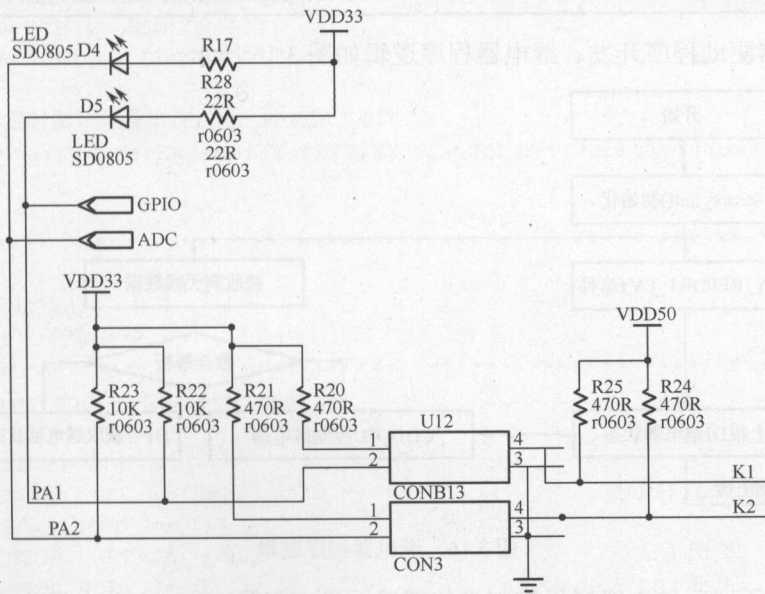


图 3.14 继电器模块与 CC2530 部分接口电路（一）

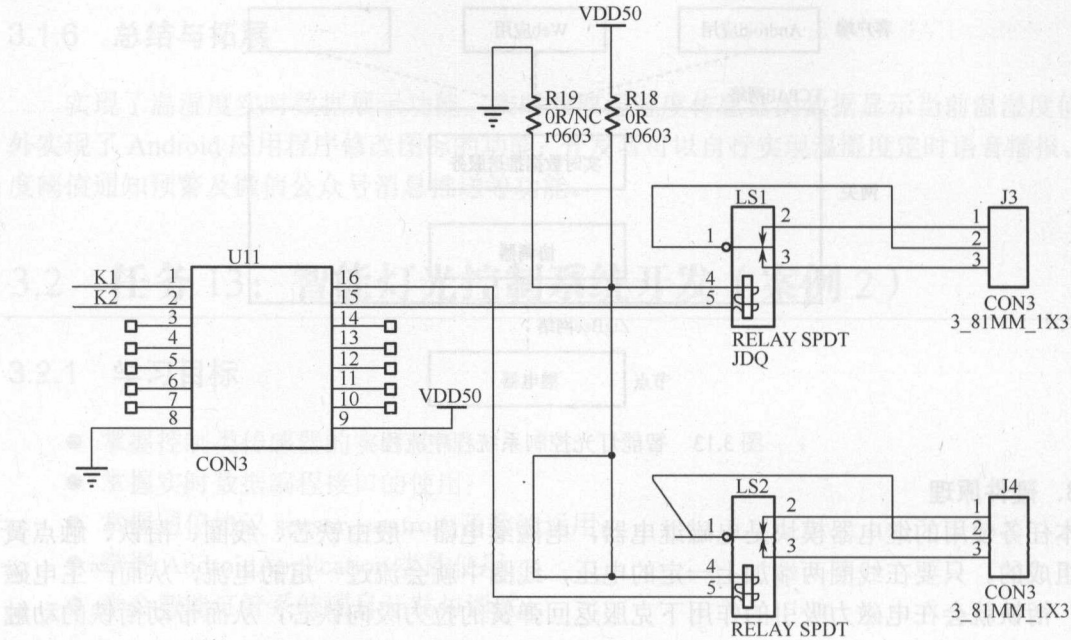


图 3.15 继电器模块与 CC2530 部分接口电路（二）

3.2.4 开发内容

1. 硬件层驱动设计

(1) ZXBee 智云数据通信协议。根据 2.1 节介绍，定义继电器的通信协议如表 3.4 所示。

表 3.4 相关传感器智云通信协议定义

传感器	属 性	参 数	权限	说 明
继电器	继电器开合	D1(OD1/CD1)	R(W)	D1 的 Bit 表示各路继电器开合状态，OD1 为开、CD1 为合

(2) 传感器驱动程序开发。继电器程序逻辑如图 3.16 所示。

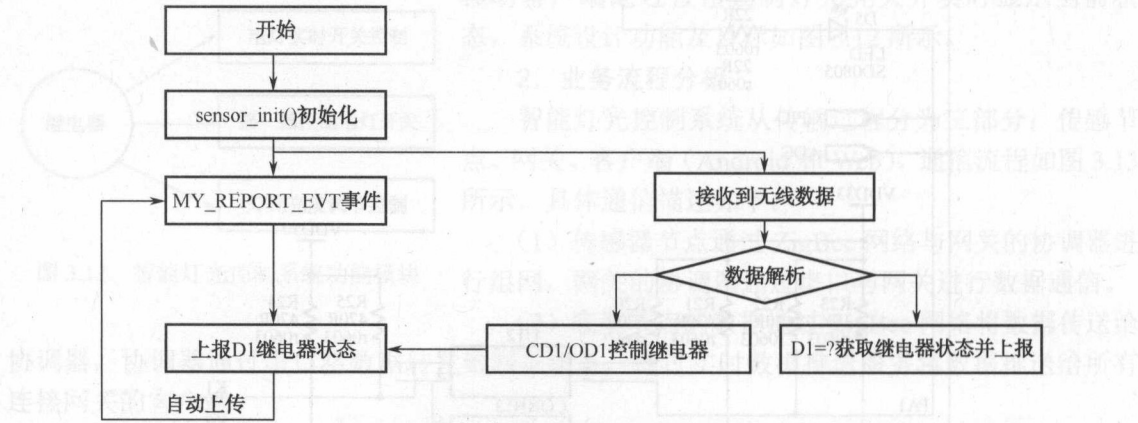


图 3.16 继电器程序逻辑

根据 2.1 节所述，继电器属于控制类传感器，设定每隔 120 s 主动上报传感器数值。相关



传感器 ZXBee HAL 函数如表 3.5 所示。

表 3.5 相关传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	初始化传感器
sensor_control()	传感器控制函数
sensor_update()	上报采集到的数据
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	自定义事件处理函数，启动定时器触发事件 MY_REPORT_EVT

部分程序代码如下。

```

/*****宏定义*****/
#define SENSOR_PORT  P0
#define SENSOR_SEL  P0SEL
#define SENSOR_DIR  P0DIR
#define SENSOR_BIT  0x22
/*****全局变量*****/
static uint8 D1 = 0;           //继电器初始状态为全关
static uint16 V0 = 120;        //V0 设置为上报时间间隔，默认为 120s
static uint16 myReportInterval = 120;    //上报时间间隔，单位为 s
/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    //配置 P0_1、P0_5 端口为通用输出 IO
    SENSOR_SEL &= ~(SENSOR_BIT);
    SENSOR_DIR |= SENSOR_BIT;
    SENSOR_PORT |= SENSOR_BIT;           //LS1/LS2 断开

    //启动定时器，触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID,MY_REPORT_EVT,(uint16)((osal_rand()%10)
                                                                    *1000));
}
/*****
*名称: sensor_control()
*功能: 传感器控制
*参数: cmd - 控制命令
*****/
void sensor_control(uint8 cmd)
{
    if (cmd == 0){
        SENSOR_PORT |= 0x22;           //LS1/LS2 断开
    } else if (cmd == 1){
        SENSOR_PORT &= ~0x02;           //LS1 闭合
        SENSOR_PORT |= 0x20;           //LS2 断开
    } else if (cmd == 2){

```




```
        SENSOR_PORT |= 0x02;                //LS1 断开
        SENSOR_PORT &= ~0x20;               //LS2 闭合
    } else if (cmd == 3){
        SENSOR_PORT &= ~0x22;               //LS1/LS2 闭合
    }
}

/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    sprintf((char*)pData, "{D1=%u}", D1);
    zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                        AF_DEFAULT_RADIUS );
    HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
}

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest{}发送给协
调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD1", ptag)) {
        D1 &= ~val;
        sensor_control(D1);
    }
    if (0 == strcmp("OD1", ptag)) {
        D1 |= val;
        sensor_control(D1);
    }
    if (0 == strcmp("D1", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D1=%u", D1);
        }
    }
    return ret;
}
```



```

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件: MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
                                                                    *1000));
    }
}

```

2. 移动端应用设计

(1) 工程框架介绍。智能灯光控制系统工程框架如表 3.6 所示。

表 3.6 智能灯光控制系统工程框架介绍

包名 (类名)	说 明
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象, 定义应用程序全局单例对象
com.zonesion.ui 子模块 Activity 包	
LightActivity.java	实时灯光控制模块

(2) 程序业务流程分析。根据应用编程接口, 智能灯光控制系统的应用设计主要采用实时数据 API 接口, 程序流程如图 3.17 所示。

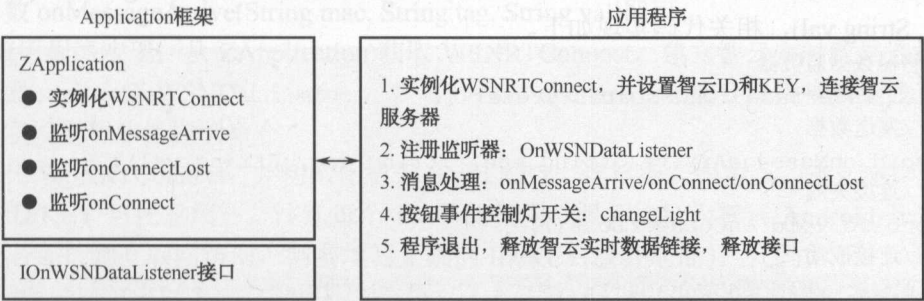


图 3.17 Android 应用程序逻辑

(3) 程序代码剖析。

① Application 框架: 实例化 WSNRTConnect 类对象并实现对实时连接的监听。

```

public class ZApplication extends Application implements WSNRTConnectListener {
    public WSNRTConnect wRTConnect; //创建 WSNRTConnect 实例

    public WSNRTConnect getWSNRConnect() {
        if (wRTConnect == null) {
            wRTConnect = new WSNRTConnect(); //初始化 WSNRTConnect 实例
        }
    }
}

```



```
        return wRTConnect;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        wRTConnect = getWSNRConnect();
        wRTConnect.setRTConnectListener(this); //设置WSNRTConnectListener 监听
    }
    //消息到达时会自动调用该方法
    @Override
    public void onMessageArrive(String mac, byte[] data) {
        .....
    }
    //连接成功
    @Override
    public void onConnect() {
        .....
    }

    //连接失败
    @Override
    public void onConnectLost(Throwable arg0) {
        .....
    }
    .....
}
```

其中，当获取到传感器上传的数据时，会调用 `onMessageArrive` 方法，若要在 `Activity` 中实现对传感器相关数据的获取，则需要在 `onMessageArrive` 方法中将消息分发，为了实现消息的分发，定义一个接口 `IONWSNDataListener`，接口中定义方法 `void onMessageArrive(String mac, String tag, String val)`，相关代码实现如下。

```
//传感器数据监听器
public interface IOnWSNDataListener {
    //发送数据
    void onMessageArrive(String mac, String tag, String val);
    //连接失败
    public void onConnectLost();
    //连接成功
    public void onConnect();
}
```

定义好接口后，在 `ZApplication` 中定义传感器数据监听数组及注册传感器监听的方法 `registerOnWSNDataListener(IOnWSNDataListener li)`，当 `Activity` 需要获取传感器数据时，需要调用 `ZApplication` 类中该方法将其加入到传感器数据监听数组中，相关代码实现如下。

```
//传感器数据监听器数组
public ArrayList<IOnWSNDataListener> mIONWSNDataListeners =
    new ArrayList<IOnWSNDataListener>();

//注册传感器数据监听器
public void registerOnWSNDataListener(IOnWSNDataListener li) {
    mIONWSNDataListeners.add(li);
}
```




```

}
//取消注册传感器数据监听器
public void unregisterOnWSNDataListener(IONWSNDataListener li) {
    mIONWSNDataListeners.remove(li);
}

```

在 `onMessageArrive` 方法中先将传感器上传的数据进行解析,然后将解析得到的数据分发给传感器数据监听器数组里面的所有监听器,让其调用接口中的 `onMessageArrive` 方法,相关代码实现如下。

```

//消息到达时会自动调用该方法
@Override
public void onMessageArrive(String mac, byte[] data) {
    if (data[0] == '{' && data[data.length - 1] == '}') {
        String sData = new String(data, 1, data.length - 2);
        String[] pDatas = sData.split(",");
        for (String pData : pDatas) {
            String[] tagVal = pData.split("=");
            if (tagVal.length == 2) {
                for (IONWSNDataListener li : mIONWSNDataListeners) {
                    //实现了 IONWSNDataListener 传感器数据监听接口的类都会
                    //自动调用 onMessageArrive() 方法
                    li.onMessageArrive(mac, tagVal[0], tagVal[1]);
                }
            }
        }
    }
}

```

以上为传感器数据的分发处理过程,所有需要获取传感器数据的 Activity 只需调用 `ZApplication` 类的 `registerOnWSNDataListener(IONWSNDataListener li)` 方法并覆写接口中的数据处理函数 `onMessageArrive(String mac, String tag, String val)` 即可。

② 应用程序实现:从 `ZApplication` 获取 `WSNRTConnect`,建立智云实时数据连接,注册数据监听器 `registerOnWSNDataListener`,复写 `onMessageArrive` 方法处理接收到无线数据,通过按钮开发发送传感器控制命令。

实例化 `WSNRTConnect`,并初始化智云 ID 和 KEY,连接智云服务器,建立连接(局域网内使用:ID/KEY 可任意填写,注意每个 ID 不能相同,服务地址为智云 Android 开发平台的 IP 地址。若连接到互联网访问,需要填写正确的 ID/KEY/服务地址),代码如下。

```

private String mMac = "00:12:4B:00:03:A7:E1:17"; //智能灯光模块 MAC 地址
private String ID = "12345678"; //用户账号
private String KEY = "12345678"; //用户密钥
mApplication = (ZApplication) getApplication(); //创建 ZApplication 实例
wRTConnect = mApplication.getWSNRTConnect(); //创建 wRTConnect 连接
wRTConnect.setIdKey(ID, KEY); //设置用户 ID 和密钥
wRTConnect.setServerAddr("zhiyun360.com:28081"); //设置智云服务地址
mApplication.registerOnWSNDataListener(this); //注册监听
wRTConnect.connect(); //连接智云服务器

```

注册传感器数据监听器。

```
mApplication.registerOnWSNDataListener(this); //注册监听
```

监听服务是否连接成功,成功则发送命令查询当前灯光状态。



```
@Override
public void onConnect() {
    //TODO Auto-generated method stub
    wRTConnect.sendMessage(mMac, "{D1=?}".getBytes()); //发送继电器状态查询命令
}
```

处理接收到的继电器控制信息。

```
@Override
public void onMessageArrive(String mac, String tag, String val) {
    //TODO Auto-generated method stub
    System.out.println("解析数据");
    if (mac.equalsIgnoreCase(mMac)) { //判断MAC地址
        if (tag.equals("D1")) { //判断参数D1
            int v = Integer.parseInt(val);
            if ((v & 0x01) == 0x01) { //若继电器状态为 on, 设置显示界面并置标志位
                layout.setBackgroundResource(R.drawable.lighton);
                changeBtn.setBackgroundResource(R.drawable.on);
                flag = "on";
            } else if ((v & 0x01) != 0x01) { //若继电器状态为 off, 设置显示界面并置标志位
                layout.setBackgroundResource(R.drawable.lightoff);
                changeBtn.setBackgroundResource(R.drawable.off);
                flag = "off";
            }
        }
    }
}
```

单击“开关”按钮控制电灯开关。

//灯光开关按钮事件响应函数, 单击按钮执行控制灯光操作

```
@SuppressWarnings("NewApi")
public void changeLight(View v) {
    if (flag.equals("on")) {
        command = "{CD1=1,D1=?}"; //若继电器当前状态为 on, 则关闭 LED1
    } else {
        command = "{OD1=1,D1=?}"; //若当前继电器状态为 off, 则打开 LED1
    }
    wRTConnect.sendMessage(mMac, command.getBytes()); //发送查询命令
    Toast.makeText(LightActivity.this, command, Toast.LENGTH_LONG).show();
}
```

③ 程序退出, 释放智云实时数据连接, 释放接口。

```
@Override
public void onDestroy() {
    wRTConnect.disconnect();
    mApplication.unregisterOnWSNDataListener(this);
    super.onDestroy();
}
```

3. Web 端应用设计

根据智云 Web 应用编程接口定义, 智能灯光控制系统的应用设计主要采用实时数据 Api 接口, JS 部分控制采集代码如下。

```
<script type="text/javascript">
var myZCloudID = "12345678"; //序列号
```



```

var myZCloudKey = "12345678"; //密钥
var mySensorMac = "00:12:4B:00:03:A7:E1:17"; //传感器的MAC地址
var rtc = new WSNRTCConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
$(function() {
    rtc.setServerAddr("zhiyun360.com:28080");
    rtc.connect(); //数据推送服务连接
    $("#ConnectState").text("数据服务连接中...");
    rtc.onConnect = function() { //连接成功回调函数
        rtc.sendMessage(mySensorMac, "{D1=?}"); //向传感器发送数据
        $("#ConnectState").text("数据服务连接成功!");
    };
    rtc.onConnectLost = function() { //数据服务掉线回调函数
        $("#ConnectState").text("数据服务掉线!");
    };
    rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
        console.log(mac, " >>> ", dat);
        if (mac != mySensorMac) { //判断传感器MAC地址
            console.log("'" + mac + " not in sensors");
            return;
        }
        if (dat[0] == '{' && dat[dat.length - 1] == '}') { //判断字符串首尾是否为{}
            dat = dat.substr(1, dat.length - 2); //截取{}内的字符串
            var its = dat.split(','); //以','来分割字符串
            for (var x in its) {
                var t = its[x].split('='); //以'='来分割字符串
                if (t.length != 2) continue;
                if (t[0] == "D1") { //判断参数D1
                    var LightStatus = parseInt(t[1]);
                    var anNiu = document.getElementById("button");
                    var bG = document.getElementById("bg");
                    if (LightStatus) {
                        anNiu.src = ("images/jdq-an-on.png");
                        bG.src = ("images/jdq-on.png");
                    } else {
                        anNiu.src = ("images/jdq-an-off.png");
                        bG.src = ("images/jdq-off.png");
                    }
                }
            }
        }
    };
});
var flag = true;
function anniu() {
    if (flag) {
        rtc.sendMessage(mySensorMac, "{CD1=1,D1=?}"); //向传感器发送数据
    } else {
        rtc.sendMessage(mySensorMac, "{OD1=1,D1=?}"); //向传感器发送数据
    }
}

```




```
flag = !flag  
};  
</script>
```

3.2.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个继电器传感器无线节点, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到 “C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples” 文件夹下。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 LightDemo 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入智能灯光控制模块界面, 在主界面弹出“连接网关成功”消息, 表示链接到智云服务中心, 通过屏幕左边的“开关”按钮控制电灯开关, 如图 3.18 所示。



图 3.18 Android 端智能灯光控制系统展示

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址 (若在局域网内使用, 则设置为智云 Android 开发平台的 IP) 和智云 ID/KEY。

(2) 将计算机接入互联网, 或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器 (或支持 HTML5 技术的 IE10 以上版本浏览器) 运行 Web 工程 “LightDemo-Web\LightDemo.html”, 进入智能灯光控制模块界面, 在主界面右上角显示“数据服务连接成功!”消息, 表示链接到智云服务中心, 通过屏幕左边的“开关”按钮控制电灯开关, 如图 3.19 所示。

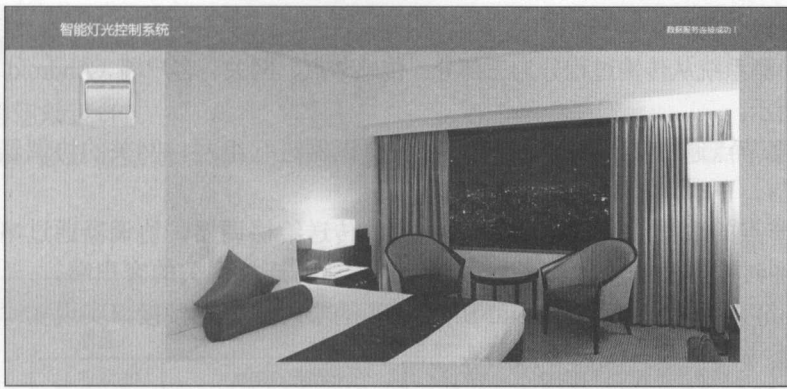


图 3.19 Web 端智能灯光控制系统展示

3.2.6 总结与拓展

本任务通过对继电器的控制，实现了远程智能控制电灯开关的功能，在此基础上可以自行实现摇一摇控制电灯开关及灯光亮度调节控制等功能。

3.3 任务 14：厨房燃气检测系统开发（案例 3）

3.3.1 学习目标

- 掌握报警类传感器的实时监测；
- 掌握实时数据编程接口的使用；
- 掌握通信协议 sensor_check()函数的运用；
- 掌握 Andorid 开发之横屏设置、ID/KEY 登录设计及 Activity 跳转与数据推送；
- 学会厨房燃气检测项目开发与调试。

3.3.2 开发环境

硬件：可燃气体传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

3.3.3 原理学习

1. 系统设计目标

通过实时监测厨房燃气报警状态，实现报警类传感器的设计，能够在 Android 移动客户端进行预警等功能，系统设计功能及目标如图 3.20 所示。

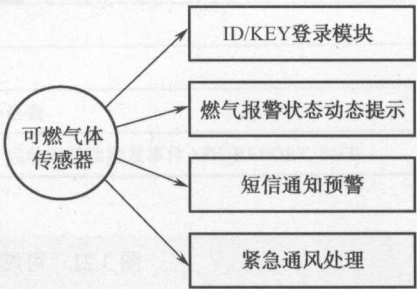


图 3.20 厨房燃气检测系统功能模块



2. 业务流程分析

厨房燃气检测系统从传输过程分为三部分：传感节点、网关、客户端（Android 和 Web），具体通信描述如下所示。

（1）传感器节点通过 ZigBee 网络与网关的协调器进行组网，网关的协调器通过串口与网关进行数据通信。

（2）底层节点的数据通过 ZigBee 网络将数据传送给协调器，协调器通过串口将数据转发给网关服务，通过实时数据推送服务将数据推送给所有连接网关的客户端。

（3）Android 应用通过调用 ZCloud SDK API 的实时数据连接接口实现实时数据采集的功能，详细实现参考下一节开发内容代码实现部分。

3. 硬件原理

本任务采用 MQ-2 型可燃气体/烟雾传感器，MQ 系列气体传感器的敏感材料是活性很高的金属氧化物粉料添加少量的铂催化剂、激活剂及其他添加剂，按一定比例烧结而成的半导体器件。最常见的如二氧化锡（ SnO_2 ）金属氧化物半导体在空中被加热到一定温度时，氧原子被吸附在带负电荷的半导体表面。半导体表面的电子会转移到吸附氧上，氧原子就变成了氧负离子，同时在半导体表面形成一个正的空间电荷层，导致表面势垒升高，从而阻碍电子流动。在工作条件下当传感器遇到还原性气体，氧负离子与还原性气体发生氧化还原反应而导致其表面浓度降低，势垒随之降低，导致传感器的阻值减小。

由二氧化锡半导体气敏材料构成，MQ-2 气体传感器所使用的气敏材料是在清洁空气中电导率较低的 SnO_2 ，当传感器所处环境中存在可燃气体时，传感器的电导率随空气中可燃气体浓度的增加而增大。MQ-2 能够有效检测甲烷、一氧化碳、氢气等可燃气体，具有检测范围广、测量灵敏度高、寿命长、驱动电路简单等优点。

使用简单的电路即可将电导率的变化转换为与该气体浓度相对应的输出信号。根据 CC2530 开发板的电路原理图得知，图 3.21 所示的 ADC 口连接到 CC2530 的 P0_1 口，当有可燃气体存在时，传感器中的电阻会发生变化，从而改变电阻处的电压值，通过 ADC 来读取可燃气体传感器电阻处实时的电压值，这样来实现实时检测。

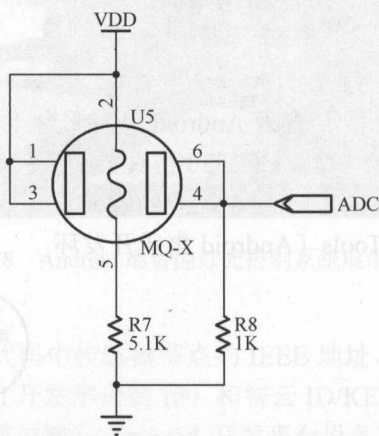


图 3.21 可燃气体传感器与 CC2530 开发板接口原理

3.3.4 开发内容

1. 硬件层驱动设计

(1) ZXBee 智云数据通信协议。根据 2.1 节介绍，可燃气体传感器的通信协议定义如表 3.7 所示。

表 3.7 相关传感器智云通信协议定义

传 感 器	属 性	参 数	权 限	说 明
可燃气体	数值	A0	R	可燃气体浓度状态， 0 或 1
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态

(2) 传感器驱动程序开发。厨房燃气检测系统程序逻辑如下。根据 2.1 节所述，可燃气体传感器属于报警类传感器，正常态每隔 120 s 主动上报传感器数值。当可燃气体浓度值达到报警阈值时，会每隔 3s 上报传感器数值，逻辑如图 3.22 所示，传感器驱动函数函数如表 3.8 所示。

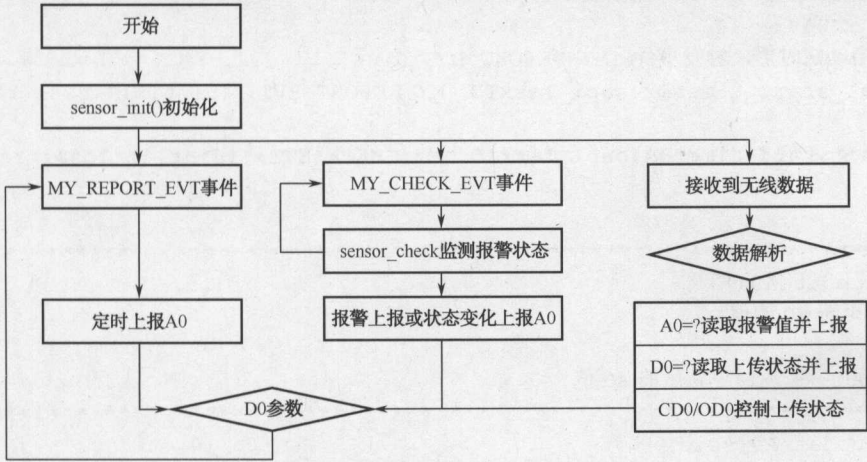


图 3.22 厨房燃气检测程序逻辑

表 3.8 相关传感器 ZXBee HAL 函数

函 数 名 称	函 数 说 明
sensor_init()	初始化传感器最基本的是配置选择寄存器和方向寄存器
updateA0()	更新传感器值
sensor_update()	上报采集到的数据
sensor_check()	监测报警状态
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	自定义事件处理函数，启动定时器触发事件 MY_REPORT_EVT

部分程序代码如下。

```
#define SENSOR_PORT P0
#define SENSOR_SEL POSEL
#define SENSOR_DIR PODIR
#define SENSOR_BIT 0x02
#define SENSOR_PIN P0_1
```



```

/*****全局变量*****/
static uint8 D0 = 1; //默认打开主动上报功能
static uint8 A0 = 0; //报警状态值
static uint16 myReportInterval = 120; //上报时间间隔, 单位为 s
static uint16 threshold = 100; //设置燃气浓度报警阈值
static uint16 Flag = 0; //报警标识
/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    //配置 P0_1 端口为输入, 且配置为外设功能 ADC:A1
    SENSOR_SEL |= SENSOR_BIT;
    SENSOR_DIR &= ~(SENSOR_BIT);

    //启动定时器, 触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10
                                                                *1000));
    osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, (uint16)((osal_rand()%10
                                                                *1000));
}
/*****
*名称: updateA0()
*功能: 更新 A0 的值
*参数: 无
*返回: A0 -- 返回更新后的 A0 值
*****/
float updateA0(void)
{
    uint16 adcValue;

    //读取 ADC:A1 采集的电压量
    adcValue = HalAdcRead(HAL_ADC_CHN_AIN1, HAL_ADC_RESOLUTION_8);
    if (adcValue > threshold){
        A0 = 1;
    } else {
        A0 = 0;
    }
    return A0;
}
/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;

```



```

uint8 pData[128];
uint8 *p = pData + 1;
int len;
//根据 D0 的位状态判定需要主动上报的数值
if ((D0 & 0x01) == 0x01){ //若报警值上报允许, 则 pData 的数据包中添加报警值数据
    updateA0();
    len = sprintf((char*)p, "A0=%u", A0);
    p += len;
    *p++ = ',';
}
//将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
if (p - pData > 1) {
    pData[0] = '{';
    p[0] = 0;
    p[-1] = '}';

    zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                                                                AF_DEFAULT_RADIUS );
    HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
}
}
/*****
*名称: sensor_check()
*功能: 监测报警值
*****/
void sensor_check(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    int len;
    uint8 lastA0 = 0;

    if((D0 & 0x01) == 1){
        lastA0 = A0; //记录上次 A0 的值
        updateA0();
        //当监测到维持高电平状态, 上报报警值 A0=1
        if (A0 == 1) {
            if(Flag % 3 == 0){ //每 3 s 报警一次
                len = sprintf((char*)pData, "{A0=%u}", A0);
                zb_SendDataRequest(0, cmd, len, (uint8*)pData, 0,
                                    AF_ACK_REQUEST, AF_DEFAULT_RADIUS); //发送数据到协调器
                HalLedSet(HAL_LED_1, HAL_LED_MODE_BLINK); //通信 LED 闪烁一次
            }
            Flag++;
        }
        //当监测到维持低电平状态, 上报清除报警状态 A0=0
    } else if ((Flag != 0) && (lastA0 == 0) && (A0 == 0)) {
        len = sprintf((char*)pData, "{A0=%u}", A0);
        zb_SendDataRequest(0, cmd, len, (uint8*)pData, 0, AF_ACK_REQUEST,
                                                                    AF_DEFAULT_RADIUS); //发送数据到协调器
    }
}

```




```

        HalLedSet(HAL_LED_1, HAL_LED_MODE_BLINK);           //通信 LED 闪烁一次
        Flag = 0;
    }
}

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest{}发送给协
调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;
    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);
    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);
        }
    }
    if (0 == strcmp("A0", ptag)) {
        if (0 == strcmp("?", pval)) {
            updateA0();
            ret = sprintf(pout, "A0=%u", A0);
        }
    }
    return ret;
}

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件: MY_REPORT_EVT

```



```
osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
*1000));
}
if (event & MY_CHECK_EVT) {
    sensor_check();
    //启动定时器，触发事件：MY_CHECK_EVT，定时查询报警值
    osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, 1000);
}
}
```

2. 移动端应用设计

(1) 工程框架介绍：厨房燃气检测系统工程框架如表 3.9 所示。

表 3.9 厨房燃气检测系统工程框架介绍

包名（类名）	说 明
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象，定义应用程序全局单例对象
com.zonesion.ui 子模块 Activity 包	
CombustibleActivity.java	实时可燃气体浓度值显示模块
LoginActivity.java	id/key 登录模块

(2) 程序业务流程分析：根据 2.1 节智云 Android 应用编程接口定义，厨房燃气检测系统的 Android 应用程序逻辑如图 3.23 所示。

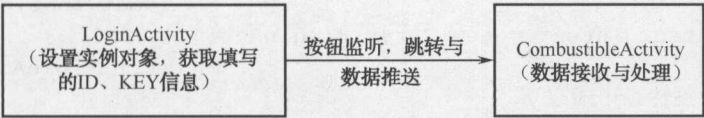


图 3.23 Android 应用程序逻辑

(3) 程序代码分析如下。

① 在 LoginActivity 中获取 EditText 控件，通过 EditText.setText()方法来设置 ID/KEY，然后通过 EditText.getText().toString()和 keyEt.getText().toString()方法获取填写的 ID/KEY。

```
public class LoginActivity extends Activity {
    private String ID = "12345678";
    private String KEY = "12345678";
    private EditText idEt;
    private EditText keyEt;
    private Button btn_antodisplay;
    private Button btn_login;
    private String idValue;
    private String keyValue;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE); //隐藏标题
        setContentView(R.layout.start);
    }
}
```



```
//EditText 和 Button 实例对象
idEt = (EditText) findViewById(R.id.id);
keyEt = (EditText) findViewById(R.id.key);
}
}
```

② 对 xml 文件中设置的两个 button 设置监听事件, 对 LOGIN 按钮设置监听跳转到 CombustibleActivity, 并传递 ID/KEY。

```
//设置“自动产生 KEY”按钮的监听
btn_antodisplay.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        idValue = idEt.getText().toString();           //获取 EditText 中的 ID 值
        //判断是否可自动产生
        if (idValue.equals(ID)) {
            keyEt.setText(KEY);
        }
    }
});
//设置 LOGIN 按钮的监听
btn_login.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        //获取两个 EditText 中的 ID、KEY 值
        idValue = idEt.getText().toString();
        keyValue = keyEt.getText().toString();
        //进行判断
        if (idValue.equals(ID) && keyValue.equals(KEY)) {
            Toast.makeText(LoginActivity.this, "登录成功",
                           Toast.LENGTH_LONG).show();
            Intent intent = new Intent(LoginActivity.this,
                                       CombustibleActivity.class);
            //跳转 Activity
            intent.setClass(LoginActivity.this, CombustibleActivity.class);
            //数据推送
            Bundle bundle = new Bundle();
            bundle.putString("idValue", idValue);
            bundle.putString("keyValue", keyValue);
            intent.putExtras(bundle);
            LoginActivity.this.startActivity(intent);
        } else {
            Toast.makeText(LoginActivity.this, "登录失败",
                           Toast.LENGTH_LONG).show();
        }
    }
});
}
```

③ 在 CombustibleActivity 中获取传递过来的 ID/KEY, 设置 ID/KEY 后进行实时连接。

```
public class CombustibleActivity extends Activity implements
                                                                    IOnWSNDDataListener {
    private ImageView alarmStatus;
    private TextView combustibleTextView;
    private AnimationDrawable recordingTransition;
```




```

private ImageButton statusControl;
private ZApplication mApplication; //ZApplication 实例化
private WSNRTConnect wRTConnect; //WSNRTConnect 实例化
private String mMac = "00:12:4B:00:03:D4:3E:F5"; //传感器 MAC 地址
//private String mMac="00:12:4B:00:02:63:3C:B7";
private int status = 0; //0 代表关闭, 1 代表开启, 2 代表报警
private String COMBUSTIBLETURE = "当前状态安全";
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE); //隐藏 Title
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
    setContentView(R.layout.combustible);
    //获取 LoginActivity 推送来的 id、key
    Intent intent = this.getIntent();
    Bundle bundle = intent.getExtras();
    String idValueGet = bundle.getString("idValue");
    String keyValueGet = bundle.getString("keyValue");
    mApplication = (ZApplication) getApplication();
    wRTConnect = mApplication.getWSNRConnect();
    mApplication.registerOnIOnWSNDataListener(this); //注册监听, 数据到, 进行处理
    //进行连接

    wRTConnect.setServerAddr("zhiyun360.com:28081");
    wRTConnect.setIdKey(idValueGet, keyValueGet);
    wRTConnect.connect();
    combustibleTextView = (TextView) findViewById(R.id.combustibleInfo);
    combustibleTextView.setText(COMBUSTIBLETURE);
    alarmStatus= (ImageView) findViewById(R.id.alarmstatus);
    statusControl = (ImageButton) findViewById(R.id.statuscontrol);
    statusControl.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            //TODO Auto-generated method stub
            switch(status)
            {
                case 0:
                    wRTConnect.sendMessage(mMac, "{D0=1}".getBytes());
                    alarmStatus.setBackgroundResource(R.drawable.on);
                    statusControl.setBackgroundResource(R.drawable.button_on);
                    status=1;
                    break;
                case 1:
                    wRTConnect.sendMessage(mMac, "{D0=0}".getBytes());
                    alarmStatus.setBackgroundResource(R.drawable.off);
                    statusControl.setBackgroundResource(R.drawable.button_off);
                    status=0;
                    break;
                case 2:
                    wRTConnect.sendMessage(mMac, "{D0=0}".getBytes());

```



```
        alarmStatus.setBackgroundResource(R.drawable.off);
        statusControl.setBackgroundResource(R.drawable.button_off);
        combustibleTextView.setText(COMBUSTIBLETURE);
        status=0;
        break;
    }
}
});
}
@Override
public void onDestroy() {
    super.onDestroy();
    wRTConnect.disconnect();
    mApplication.unregisterOnIONWSNDataListener(this);
}
@Override
public void onMessageArrive(String mac, String tag, String val) {
    if (mMac.equalsIgnoreCase(mac)) {
        if (tag.equals("A0")) { //键值对第一个等于 A0
            float v = Float.parseFloat(val); //将键值对 A0 对于的值进行格式转换
            System.out.println(val);
            if (status==1) //开启传感器
            {
                if (v == 1) {
                    combustibleTextView.setText("DANGEROUS");
                    //报警
                    alarmStatus.setBackgroundResource(R.drawable.gifmove);
                    recordingTransition=(AnimationDrawable)alarmStatus.
                                                                getBackground();
                    recordingTransition.start();
                    status=2;
                }
            }
            if(status==2) {
                if(v==0) {
                    alarmStatus.setBackgroundResource(R.drawable.on);
                    combustibleTextView.setText(COMBUSTIBLETURE);
                    status=1;
                }
            }
        }
    }
}

public void onConnectLost() {
    //TODO Auto-generated method stub
}
@Override
public void onConnect() {
    //TODO Auto-generated method stub
```



```
WRTConnect.sendMessage(mMac, "{A0=?}".getBytes());
```

```
}
```

④ 强制横屏：在 `CombustibleActivity.java` 文件中添加 `onResume()` 方法，在该方法中设置横屏。

```
protected void onResume()
{
    //设置为横屏
    if(getRequestedOrientation()!=ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE){
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
    }
    super.onResume();
}
```

3. Web 端应用设计

根据 2.1 节智云 Web 应用编程接口定义，厨房燃气检测系统的应用设计主要采用实时数据 API 接口，JS 部分控制采集代码如下。

```
<!DOCTYPE html>
<html>

<head lang="en">
<meta charset="UTF-8">
<title>
厨房燃气监测系统
</title>
<script src="js/WSNRTConnect.js" type="text/javascript">
</script>
<script src="js/jquery-1.11.0.min.js" type="text/javascript">
</script>
<style type="text/css">
body {margin: 0;padding: 0;background-color: #d9ad86;font-family: '微软雅黑',
'黑体','宋体',serif;}
.header {width: 100%;background-color: #a77a59;} h1 {padding: 0;margin:
0 0 15px;font-size: 24px;color: #fff;line-height: 3em;font-weight: 100;}
h1 small{ font-size: 12px;color: #fff;margin-left: 10px;} h1 span
{float:right;font-size:
14px;} .content {width: 1100px;margin: 0 auto;} .rq {position: absolute;top:
30px;left: 315px;width: 100px;}
</style>
<script type="text/javascript">
var myZCloudID = "12345678"; //序列号
var myZCloudKey = "12345678"; //密钥
var mySensorMac = "00:12:4B:00:02:37:7E:7A"; //传感器的 MAC 地址(可燃气体传感器)
var rtc = new WSNRTConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
rtc.connect(); //数据推送服务连接
$("#ConnectState").text("数据服务连接中...") rtc.onConnect = function() {
//连接成功回调函数
rtc.sendMessage(mySensorMac, "{A0=?}"); //向传感器发送数据
$("#ConnectState").text("数据服务连接成功!");
};
```




```

rtc.onConnectLost = function() { //数据服务掉线回调函数
    $("#ConnectState").text("数据服务掉线!");
};
rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
    if ((mac == mySensorMac) && (dat.indexOf(",") == -1)) { //接收数据过滤
        //将原始数据的数字部分分离出来
        dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf(","));
        if (parseInt(dat) > 10) {
            document.getElementById("gas").src = ("images/ranqi.gif");
        }
    }
};
</script>
</head>

<body>
<div class="header">
<div class="content">
<h1>
厨房燃气监测系统
<small>
可燃气体数值大于10发出警报
</small>
<span>
<lable id="ConnectState">
</lable>
</span>
</h1>
</div>
</div>
<div class="content" style="position:relative;">

<!-- 开 -->

<!-- 关 -->
<!--  -->
<!-- 报警 -->
<!--  -->
</div>
</body>
</html>

```

3.3.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个可燃气体传感器无线节点, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程, 将本任务 SensorHalExamples 下所有文件夹复制到 “C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples” 文件夹下。



- (3) 分别打开协调器和传感器工程，编译代码。
- (4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中，同时读取传感器节点板的 IEEE 地址。
- (5) 参考 2.1 节内容部署硬件，组成智云无线传感网络，并将数据接入到智云服务中心。

2. Android 应用程序开发

- (1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。
- (2) 编译 CombustibleDemo 工程，并安装应用程序到 Android 开发平台或 Android 终端内。
- (3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入 ID/KEY 登录界面，输入设置好的 ID，单击自动显示 KEY 按钮，在 KEY 的 EditText 中显示出设置好的 key，单击 LOGIN 按钮进入可燃气体实时数据显示界面，并显示出“登录成功”，如图 3.24 所示。

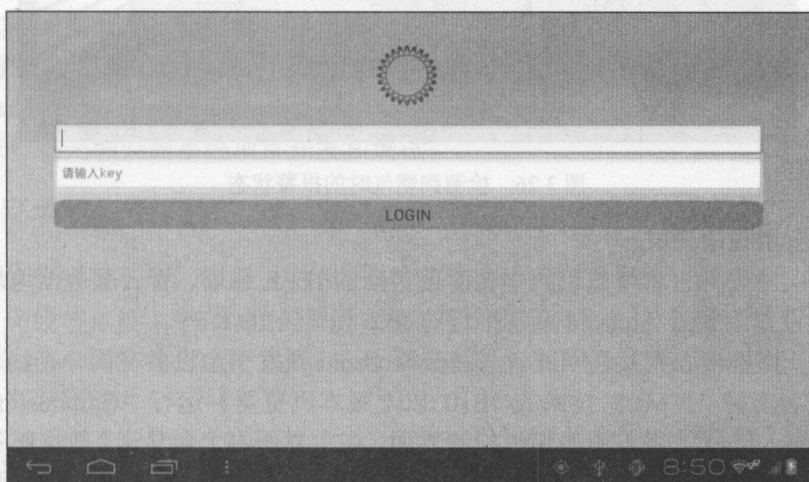


图 3.24 厨房燃气检测系统登录界面

- (4) 进入厨房燃气检测模块界面后，在主界面弹出“连接网关成功”消息后即表示链接到智云服务中心，在中间会提示当前可燃气体的状态，如图 3.25 所示。



图 3.25 厨房燃气检测模块界面



(5) 测试：正常状态下，界面会显示信息：“当前状态安全”，背景中的传感器指示灯处于暗的状态；当用打火机对传感器进行喷气测试时，指示灯闪烁，显示信息为“Dangereous”，如图 3.26 所示。



图 3.26 检测到燃气时的报警状态

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 设置计算机接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或者支持 HTML5 技术的 IE10 以上版本浏览器）运行“CombustibleDemo-Web\CombustibleDemo.html”，进入厨房燃气检测界面，在主界面右上角显示“数据服务连接成功！”消息后即表示链接到智云服务中心，并显示当前的可燃气体浓度值，如图 3.27 所示。

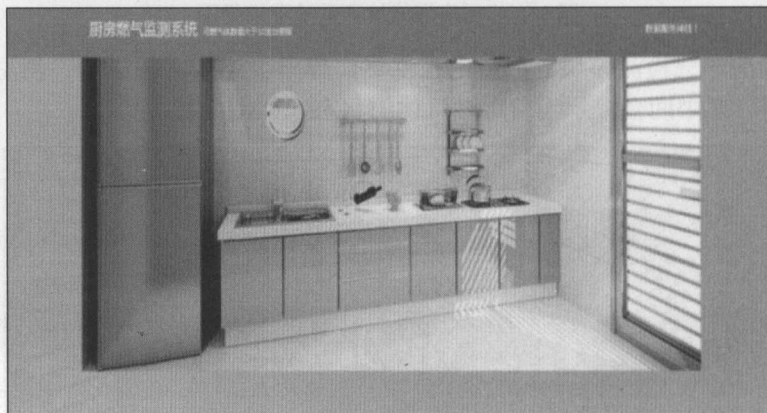


图 3.27 厨房燃气检测系统 Web 端

3.3.6 总结与拓展

厨房燃气检测系统重点实现了可燃气体浓度检测并实时显示到客户端，学习了 Android 中的对两个 Activity 的跳转与数据推送，开发者还可以自行实现的功能如下。



(1) 页面的切换同样可以用 `setContentview` 方法，开发者可以尝试并查阅资料查看页面的切换与 `Activity` 的跳转的区别。

(2) 对于强制横屏的功能，也可以在 `.java` 文件中添加 `onResume()` 方法，在该方法中进行设置，开发者可以自行学习。

(3) 开发者也可自行实现燃气阈值通知报警功能和紧急通风处理功能。

3.4 任务 15：农作物光强监测系统开发（案例 4）

3.4.1 学习目标

- 掌握传感器数据的定时上报采集；
- 掌握实时数据和历史数据编程接口的使用；
- 掌握通信协议 `sensor_update()` 函数的应用；
- 掌握 `Android UI` 之菜单开发、曲线绘制和二维码扫描功能的开发；
- 学会农作物光强监测系统项目开发和调试。

3.4.2 开发环境

硬件：光敏传感器 1 个，智云 `Android` 开发平台 1 个（默认为 `S210` 系列 `Android` 开发平台），`CC2530` 无线节点板 1 个，`CC2530` 仿真器 1 个，调试转接板 1 个。

软件：`Windows XP/7/8`，`IAR Embedded Workbench for 8051`，`Android Developer Tools`（`Android` 集成开发环境）。

3.4.3 原理学习

1. 系统设计目标

光敏传感器作为采集类的物联网传感器，能够定时采集环境光照强度值并主动上传。通过对光敏传感器的采集监控，实现农作物光强监测的设计，能够实时地将光照强度信息推送到 `Android` 移动客户端，实现随时随地远程获取农作物光强等环境信息，农作物光强监测系统设计功能及目标如图 3.28 所示。

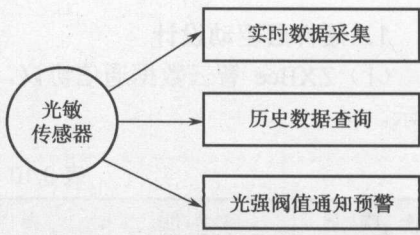


图 3.28 农作物监测系统功能模块

2. 业务流程分析

农作物光强监测从传输过程分为三部分：传感节点、网关、客户端（`Android` 和 `Web`），通信流程图如图 3.29 所示，具体通信描述如下。

(1) 传感器节点通过 `ZigBee` 网络与网关的协调器进行组网，网关的协调器通过串口与网关进行数据通信。

(2) 底层节点的数据通过 `ZigBee` 网络将数据传送给协调器，协调器通过串口将数据转发给网关服务，通过实时数据推送服务将数据推送给所有连接网关的客户端；通过历史数据存储服务将数据存储在数据中心。

(3) `Android` 应用通过调用 `ZCloud SDK API` 的实时数据连接接口实现实时数据采集的功

能，通过调用历史数据访问接口实现历史数据的查询。

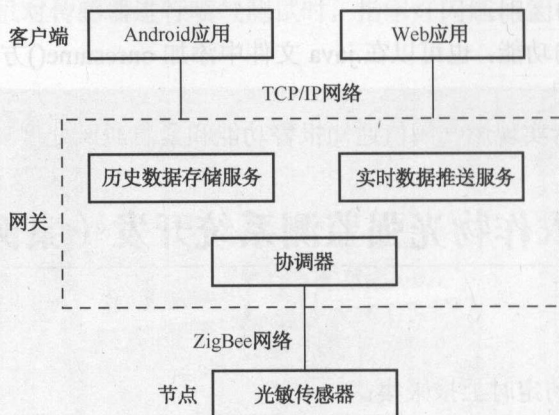


图 3.29 农作物监测业务流程

3. 硬件原理

本任务中采用的传感器为光敏传感器，它是利用光敏元件将光信号转换为电信号的传感器，其敏感波长在可见光波长附近，包括红外线波长和紫外线波长。光传感器不只局限于对光的探测，它还可以作为探测元件组成其他传感器，对许多非电量进行检测，只要将这些非电量转换为光信号的变化即可。

如图 3.30 所示，ADC 引脚连接到 CC2530 的 P0_1 口，通过此 IO 口输出的控制信号，可控制 ADC 转换得到相应数值。

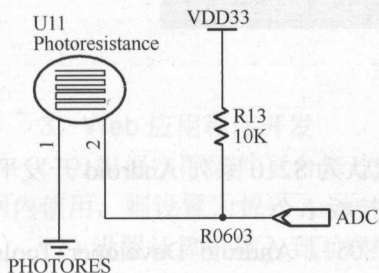


图 3.30 光敏传感器硬件原理

3.4.4 开发内容

1. 硬件层驱动设计

(1) ZXBee 智云数据通信协议。根据 2.1 节介绍，定义光敏传感器的通信协议如表 3.10 所示。

表 3.10 相关传感器智云通信协议定义

传 感 器	属 性	参 数	权 限	说 明
光敏传感器	数值	A0	R	数值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态

(2) 传感器驱动程序开发。光敏传感器程序逻辑和温湿度传感器类似，都属于采集类传感器，驱动开发设计流程如图 3.31 所示。

根据 2.1 节所述，光敏传感器属于定时采集类传感器，设定每隔 30 s 主动上报传感器数值，与温湿度传感器的工作原理类似，如表 3.11 所示。

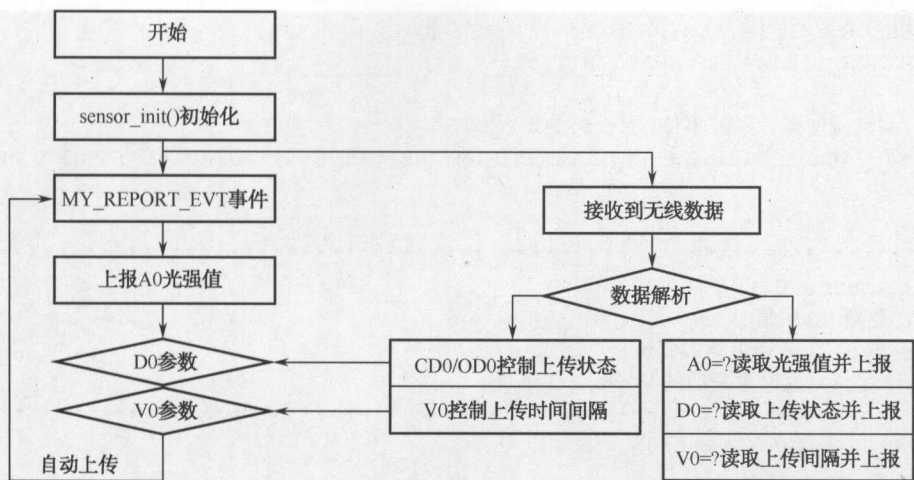


图 3.31 光敏传感器程序逻辑

表 3.11 相关传感器 ZXBee HAL 函数

核心函数名	函数说明
sensor_init()	初始化传感器最基本的是配置选择寄存器和方向寄存器
updateV0()	更新主动上报的时间间隔
updateA0()	更新传感器值
sensor_update()	上报采集到的数据
usr_process_command_call()	解析接收到的控制命令，当上层应用发送控制命令时，指定该函数进行命令解析
MyEventProcess()	自定义事件处理函数，通过该函数启动一个定时器来触发事件 MY_REPORT_EVT

部分程序代码如下。

```

/*****宏定义*****/
#define SENSOR_PORT P0
#define SENSOR_SEL POSEL
#define SENSOR_DIR PODIR
#define SENSOR_BIT 0x02
#define SENSOR_PIN P0_1

/*****全局变量*****/
static uint8 D0 = 1; //默认打开主动上报功能
static float A0 = 0; //A0 存储光照值
static uint16 V0 = 30; //V0 设置为上报时间间隔，默认为 30 s
static uint16 myReportInterval = 30; //上报时间间隔，单位为 s

*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    //配置 P0_1 端口为输入，且配置为外设功能 ADC:A1

```




```
SENSOR_SEL |= SENSOR_BIT;
SENSOR_DIR &= ~(SENSOR_BIT);

//启动定时器, 触发事件: MY_REPORT_EVT
osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10)
*1000));
}
/*****
*名称: updateV0()
*功能: 更新 V0 的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的 V0 值
*****/
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}
/*****
*名称: updateA0()
*功能: 更新 A0 的值
*参数: 无
*返回: A0 -- 返回更新后的 A0 值
*****/
float updateA0(void)
{
    uint16 adcValue;

    //读取 ADC:A1 采集的电压量
    adcValue = HalAdcRead(HAL_ADC_CHN_AIN1, HAL_ADC_RESOLUTION_8);
    A0 = (float)((1 - adcValue/256.0) * 3.3*1000) - 1680;

    return A0;
}
/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;
    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){ //若光照量上报允许, 则 pData 的数据包中添加光照量数据
        updateA0();
    }
}
```



```

    len = sprintf((char*)p, "A0=%.1f", A0);
    p += len;
    *p++ = ',';
}
//将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
if (p - pData > 1) {
    pData[0] = '{';
    p[0] = 0;
    p[-1] = '}';
    zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                                                                AF_DEFAULT_RADIUS );
    HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK );    //通信 LED 闪烁一次
}
}
/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest() 发送给协
调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;
    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);
    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);
        }
    }
    if (0 == strcmp("A0", ptag)) {
        if (0 == strcmp("?", pval)) {
            updateA0();    //更新光照量数值
            ret = sprintf(pout, "A0=%.1f", A0);
        }
    }
    if (0 == strcmp("V0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "V0=%u", V0);
        }
    }
}

```



```
        }else{
            updateV0(pval);
        }
    }
    return ret;
}
/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件: MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
                                                                    *1000));
    }
}
```

2. 移动端应用设计

(1) 工程框架介绍。农作物光强监测工程框架如表 3.12 所示。

表 3.12 农作物光强监测工程框架介绍

包名 (类名)	说 明
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象, 定义应用程序全局单例对象
com.zonesion.history 历史数据显示包	
HistoryChartActivity.java	历史数据获取与曲线图绘制类
MyDialog.java	历史数据查询等待对话框
com.zonesion.ui 子模块 Activity 包	
MainActivity.java	主模块

(2) 程序业务流程分析。相比 2.3 节中的智能灯光控制系统, 农作物光强监测系统的应用设计除了采用实时数据 API 接口外, 还采用了历史数据 API 接口, 实时数据查询的程序设计逻辑参考 2.3 节, 历史数据查询的程序实现流程如图 3.32 所示。

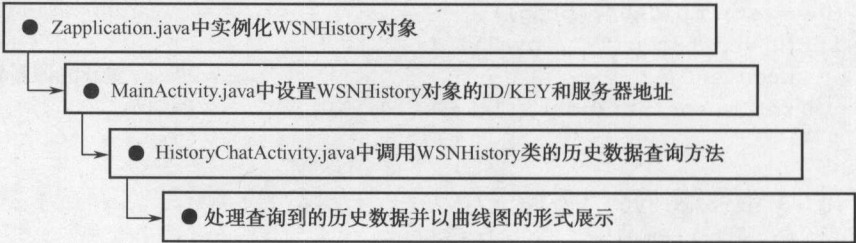


图 3.32 历史数据查询程序实现流程



(3) 程序代码剖析。

① 参照 2.3 节内容, 在 Zapplication.java 中除了实例化 WSNRTConnect 对象外, 还需实例化 WSNHistory 对象。

```
public WSNHistory getWSNHistory() {
    if (wHistory == null) {
        wHistory = new WSNHistory();           //初始化 WSNHistory 实例
    }
    return wHistory;
}
```

② 在 MainActivity.java 中设置 WSNRTConnect 对象的 ID/KEY 和服务地址。

```
public class MainActivity extends Activity implements IOnSensorDataListener {
    .....
    private WSNHistory wHistory;                //创建 WSNHistory 实例
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        .....
        wHistory = mApplication.getWSNHistory(); //获取 WSNHistory 实例
        wHistory.setIdKey(ID, KEY);              //设置 ID/KEY
        wHistory.setServerAddr("zhiyun360.com"); //设置服务器地址
    }
}
```

③ 在 MainActivity.java 中实现历史数据查询 Activity 的跳转。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    .....
    mBtn = (ImageButton) findViewById(R.id.ib_history_data);
    mBtn.setOnClickListener(new OnClickListener() {           //为按钮设置监听
        @Override
        public void onClick(View v) {
            //TODO Auto-generated method stub
            historyQuery();                                     //当单击按钮时, 调用 historyQuery() 方法
        }
    });
}

public void historyQuery() {
    Intent intent = new Intent();
    intent.putExtra("channel", "00:12:4B:00:02:CB:A9:C7_A0");
    intent.setClass(MainActivity.this, HistoryChartActivity.class);
    startActivity(intent);                                     //Activity 跳转
    //实现 activity 切换时的动画效果
    overridePendingTransition(Android.R.anim.slide_in_left,
                              Android.R.anim.slide_out_right);
}
```

④ 在 HistoryChatActivity.java 中调用 WSNHistory 类的历史数据查询方法来查询指定时间段内的数据并以曲线图的形式显示。

获取到 Zapplication.java 中实例化的 WSNHistory 对象和 MainActivity 传递过来的数据。

```
public class HistoryChartActivity extends Activity {
```



```

.....
private WSNHistory wHistory;                                //声明 WSNHistory 实例
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.historychart);

    Intent intent = getIntent();
    channel = intent.getStringExtra("channel");                //获取传递过来的数据通道
    ZApplication mApplication = (ZApplication) getApplication();
    wHistory = mApplication.getWSNHistory();                  //获取 WSNHistory 实例
}

```

调用 WSNHistory 类的 queryLast5D(String channel)方法来查询近五天的数据并对获取到的数据格式进行转换。

```

historyResult = wHistory.queryLast5D(channel);
list = getList(historyResult);

```

将转换后的数据以曲线图的形式显示。

//设置 layout 的视图, 显示 GraphicalView

```

private void showGraphicalView() {
    mGrapView = ChartFactory.getTimeChartView(getBaseContext(),
                                                buildDateDataset(x, values), renderer, "M/d-H:mm");
    LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.MATCH_PARENT);
    lp.weight = 1;
    layout.removeAllViews();
    layout.addView(mGrapView, lp);
}

```

⑤ 添加选项菜单。在 Android 系统中, 菜单可以分为三类: 选项菜单 (Option Menu)、上下文菜单 (Context Menu) 及子菜单 (Sub Menu), 本任务中使用的是选项菜单。

开发选项菜单的步骤如下。

(a) 覆写 Activity 的 onCreateOptionsMenu(Menu menu)方法, 当菜单第一次被打开时调用。

(b) 调用 Menu 的 add()方法添加菜单项(MenuItem), 同时可以调用 MenuItem 的 setIcon()方法来为菜单项设置图标。

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {                //创建选项菜单
    //TODO Auto-generated method stub
    menu.add(0, ITEM_1, 0, "设置");                            //添加菜单项 MenuItem
    menu.add(0, ITEM_2, 0, "自定义");                          //添加菜单项 MenuItem
    return true;
}

```

(c) 当菜单项(MenuItem)被选择时, 覆写 Activity 的 onOptionsItemSelected(MenuItem item)来响应事件。

```

@Override
@Override
public boolean onOptionsItemSelected(MenuItem item) {            //响应菜单单击事件
    //TODO Auto-generated method stub
    super.onOptionsItemSelected(item);
    switch (item.getItemId()) {
        case ITEM_1:

```



```

        setIdKey(id, key);           //调用 setIdKey(String id, String key) 方法
        break;
        case ITEM_2:                 //供用户自定义
            Toast.makeText(MainActivity.this, "自定义实现", Toast.LENGTH_SHORT)
                .show();
            break;
    }
    return false;
}

```

⑥ 使用二维码扫描获取 ID/KEY。Android 中集成二维码扫描功能需要使用 Google 的开源框架 Zxing，在应用中需要引用 libscAndroid.jar 包和 zxing.jar 包。libscAndroid.jar 包已经实现了二维码扫描的功能，在单击对话框中的扫描功能时，需跳转至 CaptureActivity 来实现二维码扫描，因此在程序中只需要处理扫描得到的结果即可，这里需要覆写 Activity 的 onActivityResult(int requestCode, int resultCode, Intent data) 方法。

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    //TODO Auto-generated method stub
    if (requestCode == SCAN_CODE_RESULT) {
        if (resultCode == Activity.RESULT_OK) {
            Bundle bundle = data.getExtras();
            String scanResult = bundle.getString("result");
            //解析扫描结果
            String[] tags = scanResult.split(",");
            for (String tag : tags) {
                String[] cv = tag.split(":");
                if (cv.length < 2)
                    continue;
                if (cv[0].equals("ID")) {
                    id = cv[1];
                } else if (cv[0].equals("KEY")) {
                    key = cv[1];
                }
            }
        }
    }
    setIdKey(id, key);
}

```

3. Web 端应用设计

根据智云 Web 应用编程接口定义，农作物监测的应用设计主要采用实时数据和历史数据 API 接口，js 部分代码如下。

```

$(function() {
    var myZCloudID = "123";           //序号
    var myZCloudKey = "123";          //密钥
    var mySensorMac = "00:12:4B:00:02:CB:A9:C7"; //传感器的 MAC 地址
    var channel = '00:12:4B:00:02:CB:A9:C7_A0'; //数据通道
    var myHisData = new WSNHistory(myZCloudID, myZCloudKey);
                                           //建立对象,并初始化
    var rtc = new WSNRTCConnect(myZCloudID, myZCloudKey);
                                           //创建数据连接服务对象
    rtc.connect();                     //数据推送服务连接
}

```




```
$("#ConnectState").text("数据服务连接中...");
rtc.onConnect = function() { //连接成功回调函数
    rtc.sendMessage(mySensorMac, "{A0=?}"); //向传感器发送数据
    $("#ConnectState").text("数据服务连接成功!");
};
rtc.onConnectLost = function() { //数据服务掉线回调函数
    $("#ConnectState").text("数据服务掉线!");
};
rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
    if ((mac == mySensorMac) && (dat.indexOf(",") == -1)) {
        //接收数据过滤, 将原始数据的数字部分分离出来
        dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf(","));
        $("#LightIntensity").text(dat); //显示接收到的原始数据
    }
};
...
});
```

3.4.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个光敏传感器无线节点, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 分别打开协调器和传感器工程, 编译代码。

(3) 使用 Flash Programmer 工具把上述程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(4) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 LightMonitor 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入农作物光强监测系统主界面, 在主界面弹出“连接网关成功”消息后即表示连接到智云服务中心, 在屏幕中间会实时地显示当前的光照强度值, 如图 3.33 所示。



图 3.33 光强值实时显示



(4) 单击历史数据查询按钮，会查询近五天的光照强度值，并以曲线的形式显示，如图 3.34 所示，可使用右下角的缩放按钮来查看曲线图。

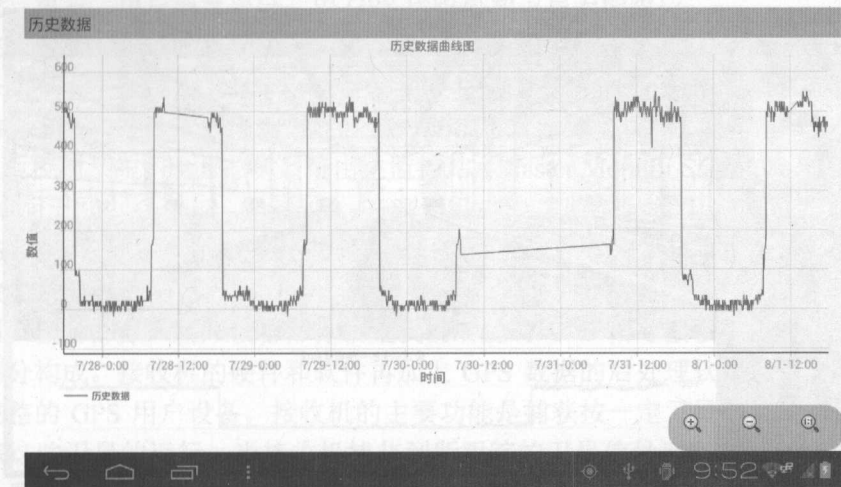


图 3.34 历史数据查询

(5) 单击设备的 Menu 按键（若没有物理 Menu 键，可单击底部的菜单按钮（三个点）），会弹出选项菜单，单击设置菜单项进入 ID/KEY 的设置，如图 3.35 所示。可以使用二维码扫描来快速获取 ID/KEY，若更改了 ID/KEY，程序会进行重连。

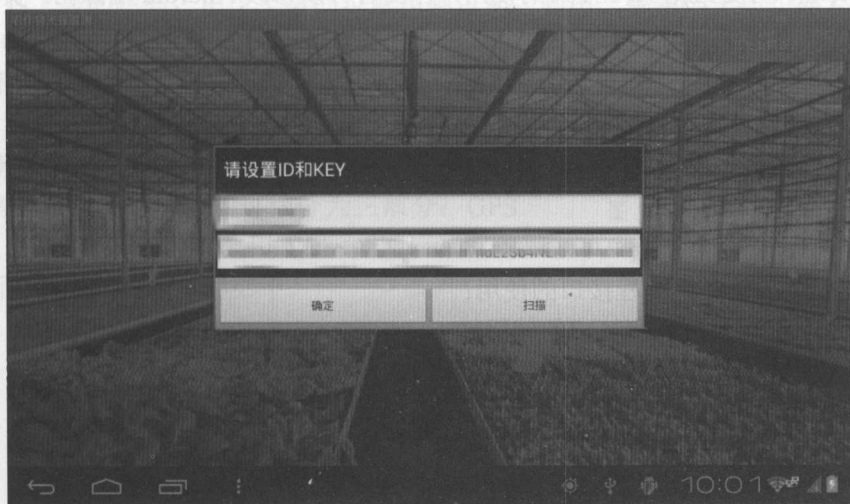


图 3.35 ID/KEY 设置

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 将计算机接入互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“LightMonitor-Web\LightMonitor.html”，进入农作物光强监测界面，在主界面右上角显示“数据服务连接成功！”



消息后即表示连接到智云服务中心，在左侧栏会实时地显示当前光强值，在页面下方会显示最近一天的光强值曲线图，如图 3.36 所示。

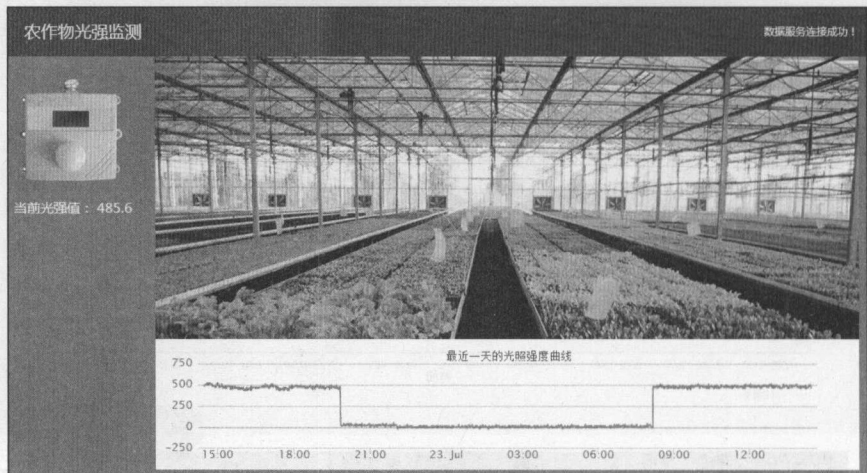


图 3.36 农作物监测网页端实现效果

3.4.6 总结与拓展

本章是基于前面章节知识点的应用及扩展，新的知识点有 Android 中选项菜单的开发和二维码扫描的实现，历史数据 API 接口的使用，开发者可以修改 Android 端源码，实现自定义选项菜单和相关的单击事件。

3.5 任务 16：GPS 网关定位系统开发（案例 5）

3.5.1 学习目标

- 掌握智云 GPS 定位编程接口的使用；
- 掌握百度地图应用程序接口的使用；
- 掌握地图覆盖物中地图标注的使用；
- 学会 GPS 网关定位系统项目开发和调试。

3.5.2 开发环境

硬件：GPS 模块（含天线）1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台）。

软件：Windows XP/7/8，Android Developer Tools（Android 集成开发环境）。

3.5.3 原理学习

1. 系统设计目标

GPS 定位系统由三部分构成。第一，空间部分：由 24 颗分布在 6 个道平面上卫星组成。



第二，地面控制部分：由地面天线（在主控站的控制下，向卫星注入寻电文）、监测站（数据自动收集中心）、主控站（负责管理、协调整个地面控制系统的工作）和通信辅助系统（数据传输）组成。第三，用户装置部分：由 GPS 接收机和卫星天线组成。

（1）空间部分：GPS 的 24 颗空间卫星，是由 21 颗工作卫星和 3 颗有源备份卫星组成，它们均匀分布在 6 个轨道面上（每个轨道面 4 颗），轨道倾角为 55° ，距地表上空 20 200 km。这种分布的卫星使得在全球任何地方、任何时间都可观测到 4 颗以上的卫星。

（2）地面控制系统：地面控制系统由主监控站（Master Monitor Station）、监测站（Monitor Station）、地面天线（Ground Antenna）三部分组成。主监控站位于美国科罗拉多州春田市（Colorado Spring），地面控制站负责收集由卫星传回的信息，并计算卫星星历、相对距离，大气校正等数据。

（3）用户设备部分：用户设备部分就是我们所说的 GPS 信号接收机，它由天线单元和接收单元两部分构成。接收机的硬件和软件再加上 GPS 数据的后处理软件包，这三样结合在一起就构成完整的 GPS 用户设备。接收机的主要功能是捕获按一定卫星截止角所选择的待测卫星，并追踪这些卫星的运行。当接收机捕获到所跟踪的卫星信号后，测量出接收天线至卫星的距离和伪距离的变化率，解调出卫星轨道参数等数据。接收机中的中央微型处理器通过这些数据，根据规定的定位解算方法，计算出用户所在地理位置的经纬度、速度、高度、时间等信息。

GPS 定位原理 GPS 的工作原理是以我们平时所熟知的几何及物理方面的基本原理为基础的。在进行 GPS 观察时，接收机一定是位于以卫星为中心、所测得距离为半径的圆球上，可得到接收机到卫星的距离，由于卫星的位置精确可知，利用三维坐标中的距离公式，根据 3 颗卫星和接收机一定处于 3 个圆环的交点上，可以列出 3 个位置方程，排除掉一个不合理的结果，就能得出准确的所在位置。系统设计功能及目标如图 3.37 所示。

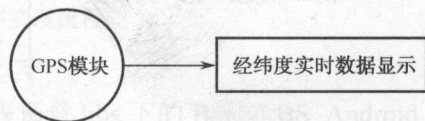


图 3.37 GPS 网关定位系统

2. 业务流程分析

GPS 网关定位系统从传输过程分为三部分：GPS 模块、网关、客户端（Android 和 Web），通信流程图如图 3.38 所示，具体通信描述如下。

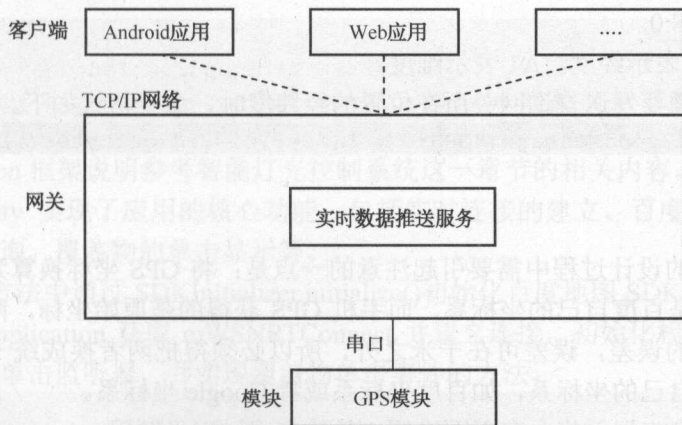


图 3.38 GPS 网关定位系统程序流程



- (1) GPS 模块通过串口与网关进行数据通信。
- (2) 网关服务通过实时数据推送服务将数据推送给所有连接网关的客户端。
- (3) Android 应用通过调用 ZCloud SDK API 的实时数据连接接口实现经纬度数据采集的功能，详细实现参考下一节开发内容代码实现部分。

3. 硬件原理

本任务中采用了 SkyTraq 公司的 GPS 接收模块 S1315R。S1315R 的输出信号是根据 NMEA (National Marine Electronics Association) 0183 格式标准输出的，输出信息主要包括位置测定系统定位资料 GPGGA、偏差信息和卫星状态 GPGSA、导航系统卫星相关资料 GPGSV、最起码的 GNSS 信息 GPRMC，共 4 个部分，电路图如图 3.39 所示。

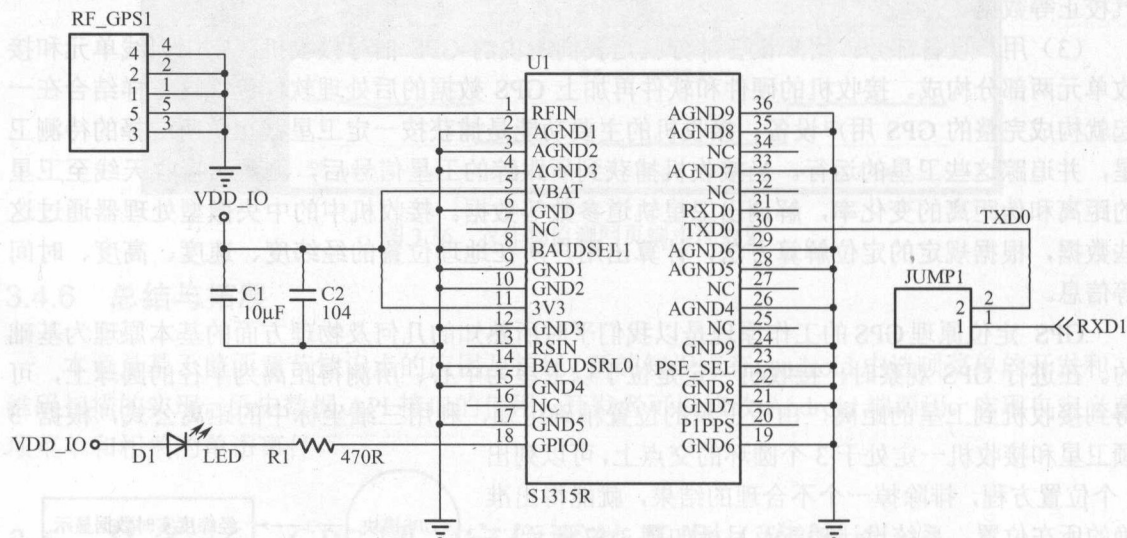


图 3.39 GPS 模块硬件原理

4. 原理学习

GPS 服务采用智云实时连接接口，GPS 模块作为一个虚拟节点向上层应用输出网关的经纬度信息，协议定义如下。

- (1) 地址：GPS:0。
- (2) 参数：A0 表示经度，A1 表示维度。
- (3) 示例：当需要发送查询网关所在位置的经纬度时，命令格式如下。

```
mWSNRTConnect.sendMessage("GPS:0", "{A0=?,A1=?}".getBytes());
```

3.5.4 开发内容

GPS 定位系统的设计过程中需要引起注意的一点是：将 GPS 坐标换算为百度坐标，百度对外提供的坐标系是百度自己的坐标系，而手机 GPS 获得的是原始坐标，两者不在一个坐标系上，所以有很大的误差，误差可在千米之外，所以必须得把两者换成统一坐标系。可以统一各个地图运营商自己的坐标系，如百度坐标系或者 Google 坐标系。

如何转换成百度坐标系将会在下面讲解，具体实现过程见源码。



1. 移动端应用设计

(1) 工程框架介绍。GPS 网关定位系统工程框架如表 3.13 所示。

表 3.13 GPS 网关定位系统工程框架介绍

包名（类名）	说 明
com.zonesion.gpslocation.application 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象，定义应用程序全局单例对象
com.zonesion.gpslocationactivity 包	
MainActivity.java	主 Activity

(2) 程序业务流程分析。GPS 网关定位系统调用的是实时数据 API 接口，只是加入了百度地图的应用，程序的实现流程如图 3.40 所示。

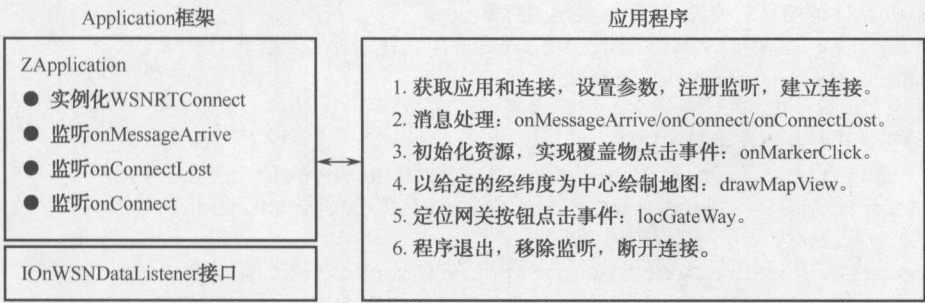


图 3.40 GPS 网关定位系统程序实现流程

(3) 程序代码剖析。

① GPS 网关定位系统使用了百度地图 SDK，即工程目录 libs 下的 BaiduLBS_Android.jar 包，在使用百度地图 SDK 提供的各种 LBS 能力之前，需要获取百度地图移动版的开发密钥，该密钥与开发者的百度账户相关联。注册和申请密钥，可以参考 <http://developer.baidu.com/>和 <http://lbsyun.baidu.com/>。申请密钥成功后，在 AndroidManifest.xml 清单文件中做相应的修改，如下所示。

```
<meta-data
    Android:name="com.baidu.lbsapi.API_KEY"
    Android:value="awzoGTpYgcwTQmFmObrgDbGT" /><!-- 此处的值要修改为用户申请的
API_KEY -->
```

② ZApplication 框架说明参考智能灯光控制系统这一章节的相关内容。

③ MainActivity 实现了应用的核心功能，包括实时连接的建立、百度地图的展示、GPS 定位坐标的单击查询、覆盖物的单击显示等。

在 onCreate()方法中通过 SDKInitializer.initialize()初始化百度地图 SDK 引用的 Context 全局变量，通过 mApplication 获取 mWSNRTConnect 并建立连接，初始化相关资源控件，为百度地图设置覆盖物单击监听器，并实现覆盖物单击事件的方法。

```
@Override
publicvoid onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```




```
//初始化百度地图 SDK 引用的 Context 全局变量
SDKInitializer.initialize(getApplicationContext());
setContentView(R.layout.activity_main);

mApplication = (ZApplication) getApplication();           //获取应用
mWSNRTConnect = mApplication.getWSNRTConnect();          //获取连接
mWSNRTConnect.setIdKey(ID, KEY);                          //设置用户 ID 和密钥
mWSNRTConnect.setServerAddr("zhiyun360.com:28081");      //设置智云服务地址
mApplication.registerOnSensorDataListener(this);         //注册监听
mWSNRTConnect.connect();                                  //建立连接

//初始化位图资源
bdMarker01 = BitmapDescriptorFactory
    .fromResource(R.drawable.icon_marker01);
bdMarker02 = BitmapDescriptorFactory
    .fromResource(R.drawable.icon_marker02);
//初始化地图显示控件, 获取百度地图对象
mMapView = (MapView) findViewById(R.id.mvRight);
mBaiduMap = mMapView.getMap();

//初始化经纬度文本显示控件
tvLongitude = (TextView) findViewById(R.id.tvLongitude);
tvLatitude = (TextView) findViewById(R.id.tvLatitude);
//tvGateWay 显示网关信息
tvGateWay = new TextView(getApplicationContext());
tvGateWay.setBackgroundColor(Color.WHITE);
tvGateWay.setTextColor(Color.BLACK);
//为百度地图设置覆盖物单击监听器
mBaiduMap.setOnMarkerClickListener(new OnMarkerClickListener() {
    //覆盖物单击事件
    @Override
    public boolean onMarkerClick(Marker marker) {
        if (marker == mMarkerGateWay) {
            tvGateWay.setText("ID:" + ID);
            LatLng latLng = marker.getPosition();
            //-50 为偏移位置, 在覆盖物上方 50 个像素处显示
            mInfoWindow = new InfoWindow(tvGateWay, latLng, -50);
            mBaiduMap.showInfoWindow(mInfoWindow);
        }
        return true;
    }
});
}
```

在销毁 Activity 时会调用的 `onDestroy()` 方法中通过 “`mApplication.unregisterOnSensorDataListener(this);`” 移除监听, 通过 “`mWSNRTConnect.disconnect();`” 断开连接。

```
//销毁 Activity 时会自动调用该方法
@Override
protected void onDestroy() {
    mApplication.unregisterOnSensorDataListener(this);    //移除监听
    mWSNRTConnect.disconnect();                          //断开连接
}
```



```
super.onDestroy();
```

在绘制地图 `drawMapView()` 方法首先需要将 GPS 模块采集的 GPS 坐标转换成百度坐标。因为出于安全方面的考虑，百度地图中所用的坐标与 GPS 坐标不一致，如果不进行转换的话，会与实际位置存在 1 km 左右的误差。

```
//绘制地图
```

```
private void drawMapView(double latitude, double longitude) {
    LatLng latlng = new LatLng(latitude, longitude);
    latlng = GPSCoord2BaiduCoord(latlng); //硬件采集的 GPS 坐标转换成百度坐标

    MapStatus mMapStatus = new MapStatus.Builder().target(latlng).zoom(17)
                                                .build();
    MapStatusUpdate mMapStatusUpdate = MapStatusUpdateFactory
                                                .newMapStatus(mMapStatus);
    mBaiduMap.setMapStatus(mMapStatusUpdate);

    mBaiduMap.clear(); //添加新的覆盖物之前清除之前所有的覆盖物
    //添加百度地图覆盖物
    ArrayList<BitmapDescriptor> giflist = new ArrayList<BitmapDescriptor>();
    giflist.add(bdMarker01);
    giflist.add(bdMarker02);
    OverlayOptions ooGateWay = new MarkerOptions().position(latlng)
                                                .icons(giflist).zIndex(0).period(20);
    mMarkerGateWay = (Marker) (mBaiduMap.addOverlay(ooGateWay));
}
```

在定位网关 `locGateWay()` 方法中发送查询经纬度的命令，并弹出查询命令的消息。

```
public void locGateWay(View view) {
    mWSNRTConnect.sendMessage(GPS_ADDRESS, "{A0=?,A1=?}".getBytes());
    Toast.makeText(this, "{A0=?,A1=?}", Toast.LENGTH_SHORT).show();
}
```

覆写接口的方法：`onConnect()` 方法只在连接成功时调用，方法体中的内容与定位网关 `locGateWay()` 中的内容是一样的，即在连接成功后立即查询一次经纬度值而不是一直等待底层主动上传数据。

```
@Override
public void onConnect() { //连接成功时发送查询命令
    mWSNRTConnect.sendMessage(GPS_ADDRESS, "{A0=?,A1=?}".getBytes());
    Toast.makeText(this, "{A0=?,A1=?}", Toast.LENGTH_SHORT).show();
}
```

在 `onMessageArrive()` 方法中解析获取到的 GPS 坐标数据，将经纬度值显示在文本显示控件上，并以该经纬度为中心绘制地图。

```
@Override
public void onMessageArrive(String mac, String tag, String val) {
    if (GPS_ADDRESS.equalsIgnoreCase(mac)) {
        if (tag.equals("A0")) {
            tvLongitude.setText(val);
            dLongitude = Double.parseDouble(val);
        } else if (tag.equals("A1")) {
            tvLatitude.setText(val);
        }
    }
}
```



```
dLatitude = Double.parseDouble(val);
}
drawMapView(dLatitude, dLongitude);    //调用绘制地图方法
}
}
```

2. Web 端应用设计

根据 Web 应用编程接口定义, GPS 定位系统应用设计主要采用实时数据 API 接口, JS 部分控制采集代码如下。

```
<script type="text/javascript">
//百度地图 API 功能
var map = new BMap.Map("allmap");
var longitude= document.getElementById("longitude").value;    //获取经度值
var latitude= document.getElementById("latitude").value;    //获取纬度值
var point = new BMap.Point(longitude,latitude);
map.centerAndZoom(point,15 );    //初始化地图, 设置中心点坐标和地图级别
var mapType = new BMap.MapTypeControl({mapTypes:
[BMAP_NORMAL_MAP,BMAP_HYBRID_MAP]});
map.addControl(mapType);    //2D 图, 卫星图
var marker = new BMap.Marker(point);    //创建标注
map.addOverlay(marker);    //将标注添加到地图中

//缩放控件
var opts = {type: BMAP_NAVIGATION_CONTROL_ZOOM};
map.addControl(new BMap.NavigationControl(opts));
map.enableScrollWheelZoom(true);    //开启鼠标滚轮缩放

function locate_button(){
    map.removeOverlay(marker);    //移除上个标记
    longitude= document.getElementById("longitude").value;
    latitude= document.getElementById("latitude").value;
    point = new BMap.Point(longitude,latitude);
    map.centerAndZoom(point,15 );
    marker = new BMap.Marker(point);
    map.addOverlay(marker);
}

</script>
```

3.5.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个 GPS 模块, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中的智云 ID/KEY、模块的 MAC 地址、服务器地址。

(2) 编译 GPSLocation 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入 GPS 定位系统主界面，在主界面弹出“连接网关成功”消息后即表示连接到智云服务中心，输入经度和纬度，单击“定位网关”按钮，定位到当前经纬度对应的位置（网关所在位置），如图 3.41 所示。



图 3.41 Android 端 GPS 定位实现

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）、GPS 模块的 MAC 地址以及智云 ID/KEY。

(2) 将计算机接入互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“GPSLocation-Web\GPSLocation.html”，进入 GPS 网关定位系统模块界面，在主界面左上角显示“数据服务连接成功！”消息后即表示链接到智云服务中心，在左边栏显示程序中默认的经纬度，右侧会定位到当前经纬度对应的位置（网关所在位置），如图 3.42 所示。



图 3.42 Web 端 GPS 定位实现



3.5.6 总结与拓展

本任务实现了 GPS 定位的功能，并学习了百度地图接口的使用，通过经纬度获取到其对应的位置信息并展示在百度地图上。

3.6 任务 17: GSM 短信通知系统开发（案例 6）

3.6.1 学习目标

- 掌握智云实时数据编程接口的使用；
- 掌握 GSM 模块的使用；
- 学会 GSM 短信通知系统项目开发和调试。

3.6.2 开发环境

硬件：联通 GSM 卡 1 张，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台，含有 GSM 模块）。

软件：Windows XP/7/8，Android Developer Tools（Android 集成开发环境）。

3.6.3 原理学习

1. 系统设计目标

全球移动通信 GSM（Global System For Mobile Communication）是 1992 年欧洲标准化委员会统一推出的标准，它采用数字通信技术、统一的网络标准，使通信质量得以保证，并可以开发出更多的新业务供用户使用。GSM 移动通信网的传输速度为 9.6 kbps。目前，全球的 GSM 移动用户已经超过 5 亿，覆盖了 1/12 的人口，GSM 技术在世界数字移动电话领域所占的比例已经超过 70%。由于 GSM 相对模拟移动通信技术是第二代移动通信技术，所以简称 2G。

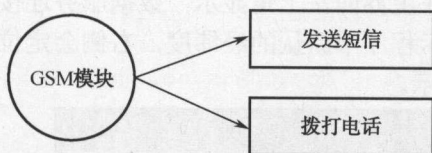


图 3.43 GSM 短信通知系统功能模块

GSM 短信通知系统设计功能及目标如图 3.43 所示。

2. 软件原理说明

GSM 服务采用智云实时连接接口，GSM 模块作为一个虚拟节点向上层应用输出信息，协议定义如下。

（1）地址：Phone:0。

（2）参数：发短信为“{Action=SendSMS,Number=10010,Content=hello word}”；打电话为“{Action=Tel,Number=10010}”。

（3）示例：当需要发送查询短信时，命令格式如下。

```
mWSNRTConnect.sendMessage("Phone:0","{Action=SendSMS,Number=10010,Content=tests_sms}".getBytes());
```



3.6.4 开发内容

上层应用（Android 端或者 Web 端）将消息发送给网关，网关会解析收到的消息并转发给 GSM 模块。因此发短信或者打电话的消息格式是有限制的，其中，发短信的消息格式定义为“{Action=SendSMS,Number=10010,Content=hello word}”；打电话的消息格式定义为“{Action=Tel,Number=10010}”。

1. 移动端应用设计

(1) 工程框架介绍。GSM 短信通知系统工程框架如表 3.14 所示。

表 3.14 GSM 短信通知系统工程框架介绍

包名（类名）	说 明
com.zonesion.app 应用包	
IOnWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象，定义应用程序全局单例对象
com.zonesion.activity 子模块 Activity 包	
MainActivity.java	主模块

(2) 程序业务流程分析。程序设计实现流程如图 3.44 所示。



图 3.44 GSM 短信发送程序实现流程

(3) 程序代码剖析。

- ① Zapplication.java 文件解析参考远程温湿度计这一章节相关内容。
- ② 在 MainActivity.java 中设置 WSNRTConnect 对象的 ID/KEY 和服务地址，参考远程温湿度计这一章节相关内容。
- ③ 在 MainActivity.java 中实现单击按钮发送消息。

```
//单击发送按钮单击事件
public void send(View view) {
    String sPhoneNum = etPhoneNum.getText().toString().trim();
    String sContent = etContent.getText().toString().trim();

    if (sPhoneNum.equals("") || sContent.equals("")) {
        Toast.makeText(this, "接收者或短信内容为空!", Toast.LENGTH_SHORT).show();
    } else {
        //电话节点
        //地址 Phone:0
        //打电话 {Action=Tel,Number=10010}
```




```
//发短信 {Action=SendSMS,Number=10010,Content=hello word}
//,=在网关解析数据会作为识别标记,用\,和\=替换
if (sContent.contains(",")) {
    sContent = sContent.replace(",", "\\,");
}
if (sContent.contains("=")) {
    sContent = sContent.replace("=", "\\=");
}

String data = "{ECHO=sms,Action=SendSMS,Number=" + sPhoneNum
              + ",Content=" + sContent + "}";
Toast.makeText(this, data, Toast.LENGTH_SHORT).show();
mWSNRTConnect.sendMessage(PHONE_ADDRESS, data.getBytes());
sWhiteBoard += "接收者:"
               + sPhoneNum
               + "\n"
               + "短信内容:"
               + etContent.getText().toString().trim()
               + "\n"
               + "发送时间:"
               + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss",
                                       Locale.CHINA).format(new Date(System
                                       .currentTimeMillis())) + "\n\n";
}
}
```

2. Web 端应用设计

根据 Web 应用编程接口定义, GSM 短信通知的应用设计主要采用实时数据 API 接口, JS 部分代码如下。

```
var aid = '123';
var key = '123';
var rtc = new WSNRTConnect(aid, key);
var connect = false;
rtc.onConnect = function(){
    connect = true;
    console.log("connect to server");
};
rtc.onConnectLost = function(){
    connect = false;
    console.log("disconnect from server");
};
rtc.onmessageArrive = function(mac, dat) {
    console.log(mac, ">>>", dat);
};
rtc.connect();
var log = "";
function send() {
    if (!connect) {
        console.log("与服务器连接断开");
        return;
    }
}
```

```
}  
var num = $("#number").val()  
var con = ($("#content").val());  
if (num.length == 0 || con.length == 0) return;  
var msg = "{Action=SendSMS,Number="+num+",Content="+con+"}"  
rtc.sendMessage("Phone:0", msg);  
var myDate = new Date();  
mytime=myDate.toLocaleTimeString();  
log = "&nbsp;&nbsp;["+mytime+"]&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" + num + "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<<<  
                                &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&"+con+"<br>" + log;  
$("##log").html(log);  
}
```

3.6.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台 (含 GSM 模块), 1 张联通 GSM 卡, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 将 GSM 卡插入开发平台的 GSM 模块中, 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中的智云 ID/KEY、服务器地址。

- (2) 编译 SMSNotification 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入 GSM 短信通知系统主界面, 在主界面弹出“连接网关成功”消息后即表示连接到智云服务中心, 程序功能实现如图 3.45 所示。

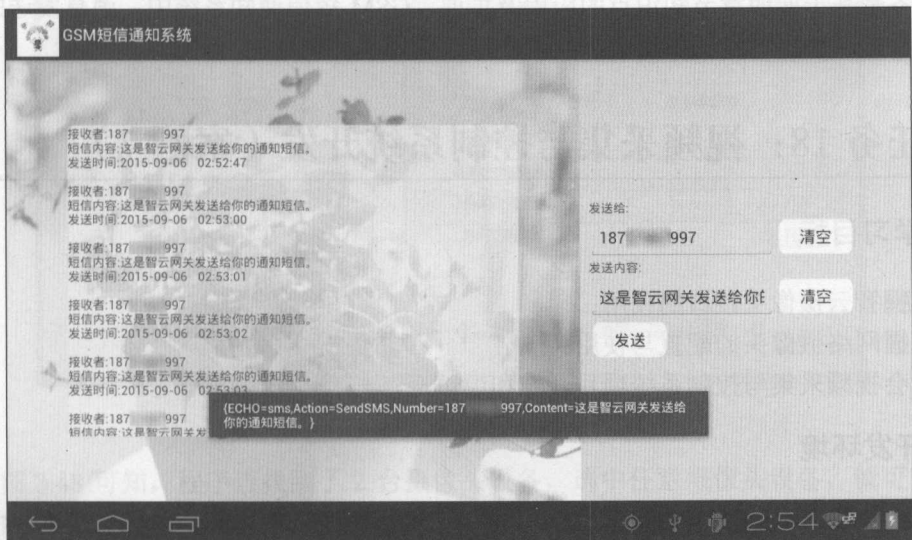


图 3.45 短信发送界面



3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中的智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 将计算机接入互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“SMSNotification-Web\SMSNotification.html”，进入 GSM 短信通知界面，如图 3.46 所示，在右侧会显示所发送的短信内容。

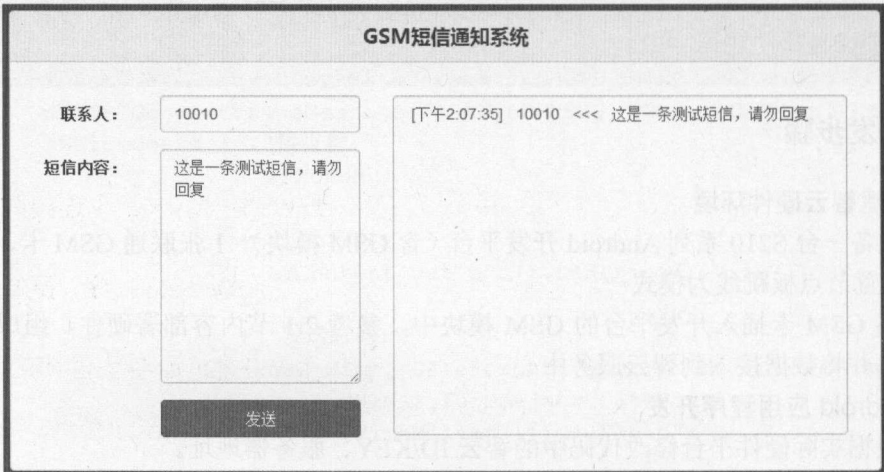


图 3.46 短信发送 Web 端实现

3.6.6 总结与拓展

本任务是基于前面章节知识点的应用及扩展，GSM 短信通知系统中，消息发送的目的地是网关，再由网关将消息转发出去，开发者可以实现拨打电话的功能。

3.7 任务 18：视频采集与控制系统开发（案例 7）

3.7.1 学习目标

- 掌握智云摄像头接口的使用；
- 掌握网络摄像头的配置与使用；
- 学会视频采集与控制系统项目开发和调试。

3.7.2 开发环境

硬件：网络摄像头一个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台）。

软件：Windows XP/7/8，Android Developer Tools（Android 集成开发环境）。



3.7.3 原理学习

1. 系统设计目标

视频采集与控制系统设计功能及目标如图 3.47 所示。

2. 业务流程分析

视频采集与控制系统的流程与 GPS 定位系统的流程相同，可参考相应章节。

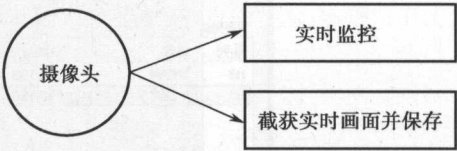


图 3.47 视频采集与控制系统功能模块

3.7.4 开发内容

网络摄像头在使用前需要先对其进行配置，摄像头的部署过程如下。

- (1) 准备 1 个 IP 摄像头，摄像头专用 5 V 电源适配器 1 个，网线一根。
- (2) 用网线的一端连接摄像头的以太网接口，另一端连接路由器；PC 也连接同一个路由器（必须用网线连接，否则查找不到摄像头），保持在同一局域网。
- (3) 查找 IP 摄像头。在“摄像头配置软件”目录下打开 FindDev.exe 程序，摄像头配置软件在配书资料目录“05-常用工具”下。
- (4) 运行上述步骤的程序后，程序就会找到相应的摄像头设备，如图 3.48 所示。

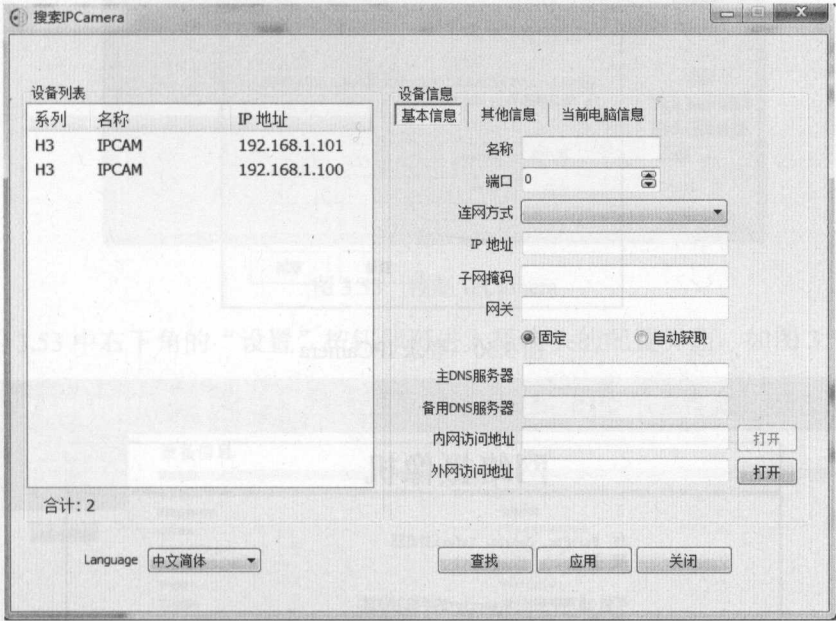


图 3.48 搜索 IPCamera

通过图 3.48 可知，程序查找到了 2 台摄像头设备，选中任意摄像头设备，就可以查看当前摄像头的 IP 地址、端口号、内网访问地址、外网访问地址等信息内容，如图 3.49 所示。

IP 地址修改说明：摄像头的 IP 地址需要设置成静态 IP，所以首先将联网方式选择为“固定的 IP 地址（静态 IP）”，然后根据 IP 地址规划自定义修改摄像头 IP 地址等信息，修改完成后，单击“应用”按钮即可，重设 IP 地址后，摄像头会重新启动。

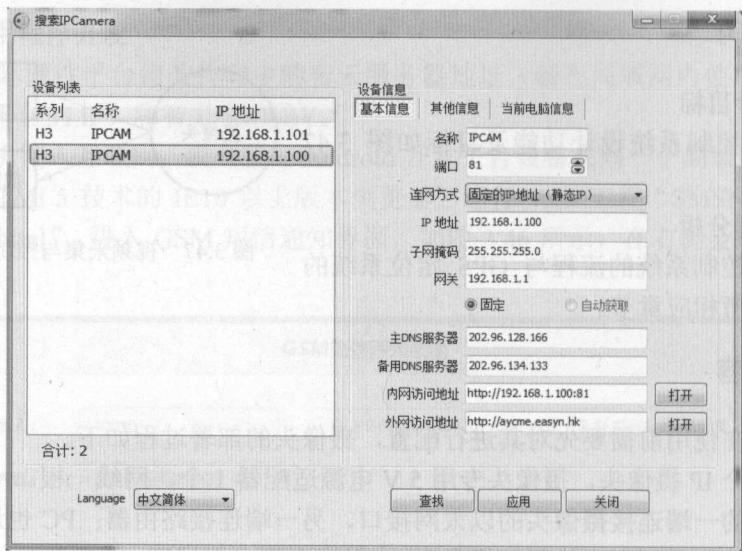


图 3.49 调看摄像头信息

(5) 查看摄像头监控画面、控制摄像头。单击上图中显示的内网访问地址“打开”按钮，按照提示输入用户名、密码（默认用户名为 admin，密码为 admin），如图 3.50 所示。

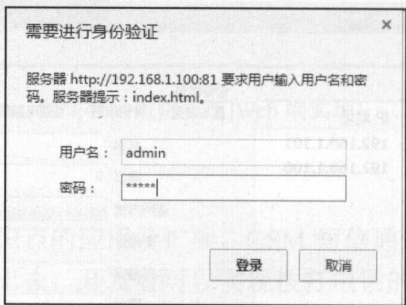


图 3.50 登录 IPCamera

登录成功之后，就会显示如图 3.51 所示的画面。

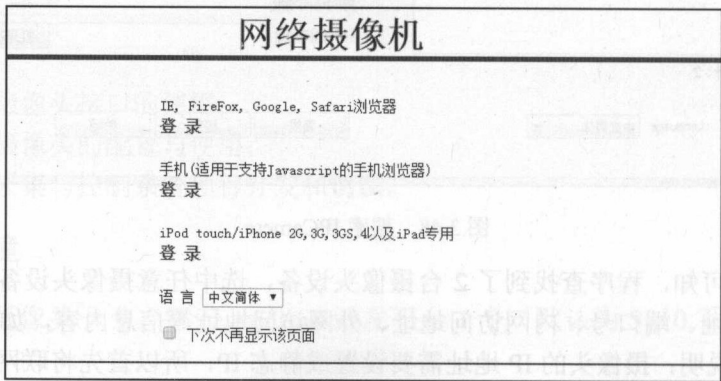


图 3.51 IPCamera 登录成功



单击第 2 个“登录”按钮，就可以看到摄像头的监控页面，如图 3.52 所示。

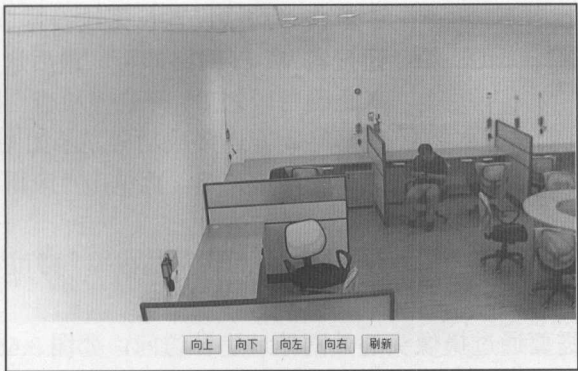


图 3.52 控制 IPCamera

若要控制摄像头则只需要单击各个控制按钮即可。

(6) 配置摄像头。在上一步骤中，登录摄像头的用户名、密码之后单击第 1 个“登录”按钮，可看到如图 3.53 所示的页面（若要查看监控页面，需要根据提示安装插件）。

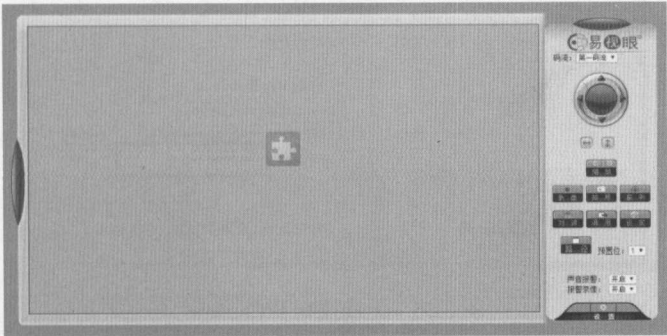


图 3.53 控制 IPCamera

单击图 3.53 中右下角的“设置”按钮即可进入摄像头的配置界面，如图 3.54 所示。

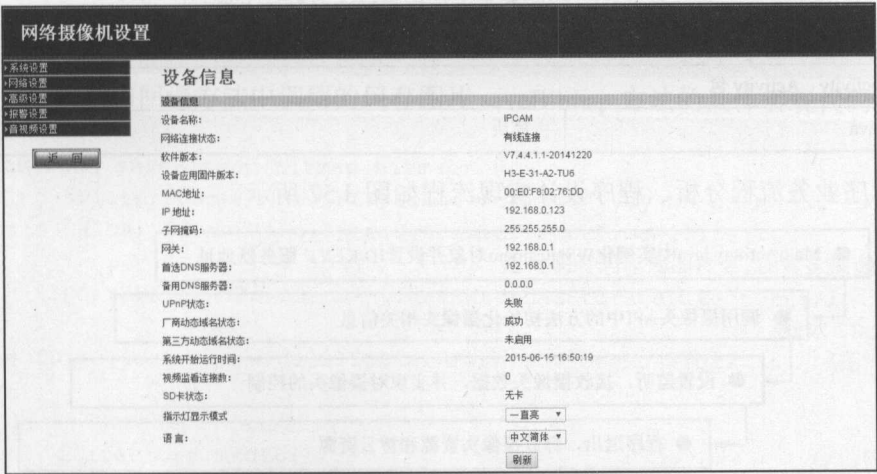


图 3.54 IPCamera 信息

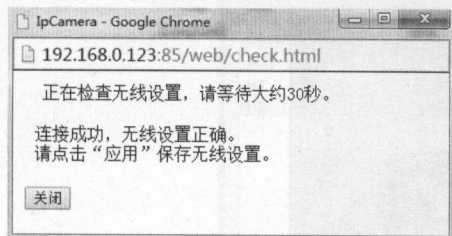


图 3.55 IPCamera 连接检测

若用户需要将摄像头连接 Wi-Fi, 则只需要在上图中单击“网络设置→无线设置”, 然后单击“搜索”按钮搜索 Wi-Fi 网络即可, 最后选中可用 Wi-Fi 网络, 输入正确密码之后, 单击“检查”按钮检查能否连接该无线网络, 若连接成功会显示如图 3.55 所示的画面。

关闭该窗口, 再单击“应用”即可(摄像头会重新启动)。

(7) 最后将网线拔掉, 即可实现无线网络访问摄像头监控页面。

(8) 如果在外网, 需要通过摄像头的外网地址进行访问, 如图 3.56 所示。

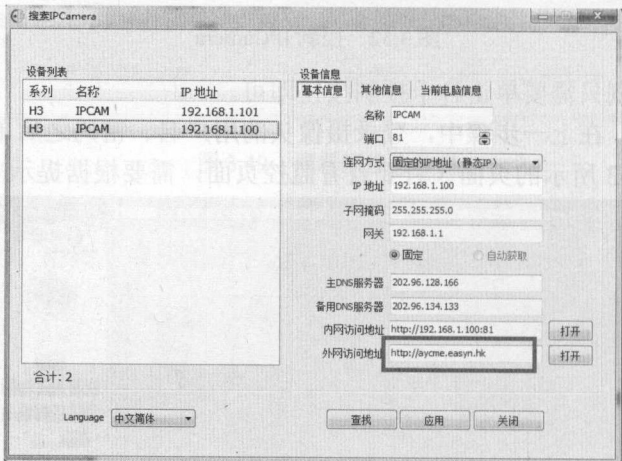


图 3.56 无线控制 IPCamera

1. 移动端应用设计

(1) 工程框架介绍。视频采集与控制系统工程框架如表 3.15 所示。

表 3.15 视频采集与控制系统工程框架介绍

包名 (类名)	说 明
com.zonesion.activity Activity 包	
MainActivity.java	主模块

(2) 程序业务流程分析。程序设计实现流程如图 3.57 所示。

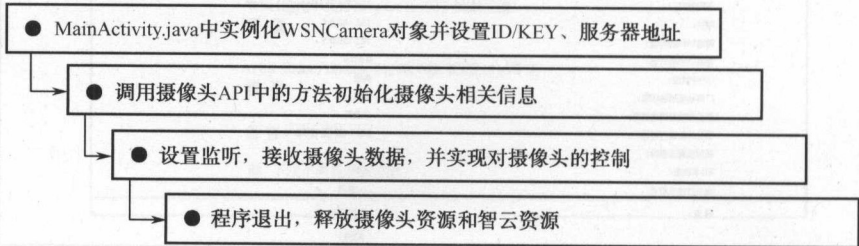


图 3.57 视频采集与控制系统实现流程



(3) 程序代码剖析。

① 在 MainActivity.java 中初始化 WSNCamera 对象并设置 ID/KEY 和服务器地址;初始化摄像头、相关信息(摄像头 IP 地址、摄像头类型、用户名、密码);设置摄像头监听。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mWSNCamera = new WSNCamera();           //创建摄像头对象
    mWSNCamera.setIdKey(ID, KEY);           //设置 ID 和 KEY
    mWSNCamera.setCameraListener(this);     //设置摄像头监听器
    mWSNCamera.initCamera(cameraIP, userName, pwd, cameraType);
                                           //初始化摄像头
    mWSNCamera.checkOnline();               //检测摄像头是否在线
}
```

② 覆写监听器里面的方法。

//摄像头检测是否在线回调方法

```
@Override
public void onOnline(String cameraIP, boolean online) {
    bOnline = online;
}
```

//摄像头抓拍回调方法

```
@Override
public void onSnapshot(String cameraIP, Bitmap bitmap) {
    saveBitmap(bitmap);           //保存图片
}
```

//视频流输出回调方法

```
@Override
public void onVideoCallBack(String cameraIP, Bitmap bitmap) {
    if (bOnOff) {
        ivVideoDisplay.setImageBitmap(bitmap);
    }
}
```

其中,摄像头抓拍回调方法中调用的保存图片 saveBitmap 方法的具体实现代码如下。

//保存图片方法

```
public boolean saveBitmap(Bitmap bitmap) {
    String sPictureName = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss",
        Locale.CHINA).format(new Date(System.currentTimeMillis()))
        + ".jpeg";           //用日期时间作为图片名,图片扩展名为 jpeg
    File fSDCardPath = Environment.getExternalStorageDirectory();
                                           //获取 SDCard 路径
    File fDirectory = new File(fSDCardPath, "VideoMonitor/"); //图片存放目录
    try {
        if (!fDirectory.exists()) {
            fDirectory.mkdir();           //创建文件目录
        }
        File file = new File(fDirectory, sPictureName);
    }
```



```

        if (!file.exists()) {
            file.createNewFile(); //创建文件
        }
        FileOutputStream out = new FileOutputStream(file); //文件输出流
        //按 90% 的图片压缩率将图片输出到文件中
        bitmap.compress(Bitmap.CompressFormat.JPEG, 90, out);
        out.flush(); //写缓冲
        out.close(); //关闭文件输出流
        Toast.makeText(this, "图片" + sPictureName + "保存在" +
            fDirectory.toString(), Toast.LENGTH_LONG).show();

        return true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}

```

③ 通过触摸屏幕来控制摄像头的代码实现如下。

```

//触摸事件
@Override
public boolean onTouch(View v, MotionEvent event) {
    float fMoveBase = 50; //手指移动距离基准为 50

    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN: //手指按下
            fPosX = event.getX(); //获取屏幕坐标 X 轴的值
            fPosY = event.getY(); //获取屏幕坐标 Y 轴的值

            dLastTime = dCurTime; //获取上次时间
            dCurTime = System.currentTimeMillis(); //获取当前时间
            if (dCurTime - dLastTime < 300) { //再次按下时间小于 300 ms 即为双击
                mWSNCamera.checkOnline();
                if (bOnline) {
                    processOnOff(); //处理开关
                } else {
                    Toast.makeText(this, "camera:" + cameraIP + "不在线",
                        Toast.LENGTH_SHORT).show();
                }
            }
            break;
        case MotionEvent.ACTION_UP: //手指拿起
            if (bOnOff) {
                float fDiffX = event.getX() - fPosX; //手指滑动水平距离
                float fDiffY = event.getY() - fPosY; //手指滑动垂直距离
                if (fDiffX > 11 * fMoveBase) {
                    mWSNCamera.control("HPATROL"); //水平巡航
                } else if (fDiffX > fMoveBase) {
                    controlCamera("LEFT", (int) (fDiffX / fMoveBase)); //向左移动
                } else if (fDiffX < -fMoveBase) {
                    controlCamera("RIGHT", (int) (fDiffX / -fMoveBase)); //向右移动
                }
            }
    }
}

```




```

        if (fDiffY > 6 *fMoveBase) {
            mWSNCamera.control("VPATROL"); //垂直巡航
        } else if (fDiffY > fMoveBase) {
            controlCamera("UP", (int) (fDiffY / fMoveBase)); //向上移动
        } else if (fDiffY < -fMoveBase) {
            controlCamera("DOWN", (int) (fDiffY / -fMoveBase)); //向下移动
        }
    }
    break;
    case MotionEvent.ACTION_MOVE: //手指移动
        break;
    default:
        break;
}
return false; //未处理完
}

```

2. Web 端应用设计

根据 Web 应用编程接口定义, 视频采集与控制系统的的核心应用设计主要采用的是智云摄像头接口, JS 部分代码如下。

```

$(function() {
    var aid = '123';
    var key = '123';

    var myImgId = "img1"; //img 标签的 ID
    var camState = 0; //初始化摄像头在线状态为离线
    var switch_cam = 0; //默认摄像头开关处于关闭状态
    var myipcamera = new WSNCamera(aid, key); //创建 myipcamera 对象
    myipcamera.setDiv(myImgId); //设置图像显示的位置

    var myCameraIP = "Camera:ayarc.easyn.hk" //摄像头 ID
    var user = "admin"; //摄像头访问用户名
    var pwd = "admin"; //摄像头访问密码
    var type = "H3-Series"; //摄像头型号
    var online = false;

    myipcamera.initCamera(myCameraIP, user, pwd, type); //摄像头初始化
    myipcamera.checkOnline(function(state) {
        if (state == 1) {
            online = true;
        }
        console.log(myCameraIP, "is online", online);
    });
    //$("#saddr").val(window.location.host+":8002");
    //截屏
    $("#imgSnapshot").click(function() {
        if (online) {
            myipcamera.snapshot();
        }
    });
});

```



```
//初始化监视器开关默认关
$("#switch").click(function() {
    var obj = $(this);
    if (!this.flag) {
        switch_cam = 1;
        if (online) {
            myipcamera.openVideo();           //打开摄像头并显示
            camState = 1;
            obj.text("关");
            obj.siblings("button").attr("class", "btn btn-success");
        }
    } else {
        switch_cam = 0;
        obj.text("开");
        obj.siblings("button").attr("class", "btn btn-default");
        myipcamera.closeVideo();             //关闭视频监控
        camState = 0;
    }
    this.flag = !this.flag;
});
//监视器控制器
$("#ct_up").mousedown(function() {           //上
    if ((switch_cam == 1) && (camState == 1)) {
        myipcamera.control("UP");             //向摄像头发送向上移动命令
        $(this).attr("class", "btn btn-info");
        $(this).mouseleave(function() {
            $(this).attr("class", "btn btn-success");
        });
        $(this).mouseup(function() {
            $(this).attr("class", "btn btn-success");
        });
    }
});

$("#ct_down").mousedown(function() {          //下
    if ((switch_cam == 1) && (camState == 1)) {
        myipcamera.control("DOWN");            //向摄像头发送向下移动命令
        $(this).attr("class", "btn btn-info");
        $(this).mouseleave(function() {
            $(this).attr("class", "btn btn-success");
        });
        $(this).mouseup(function() {
            $(this).attr("class", "btn btn-success");
        });
    }
});

$("#ct_left").mousedown(function() {           //左
    if ((switch_cam == 1) && (camState == 1)) {
        myipcamera.control("LEFT");            //向摄像头发送向左移动命令
```



```

$(this).attr("class", "btn btn-info");
$(this).mouseleave(function() {
    $(this).attr("class", "btn btn-success");
});
$(this).mouseup(function() {
    $(this).attr("class", "btn btn-success");
});
});

$("#ct_right").mousedown(function() { //右
    if ((switch_cam == 1) && (camState == 1)) {
        myipcamera.control("RIGHT"); //向摄像头发送向右移动命令
        $(this).attr("class", "btn btn-info");
        $(this).mouseleave(function() {
            $(this).attr("class", "btn btn-success");
        });
        $(this).mouseup(function() {
            $(this).attr("class", "btn btn-success");
        });
    }
});

$("#ct_h").mousedown(function() { //水平巡航
    if ((switch_cam == 1) && (camState == 1)) {
        myipcamera.control("HPATROL"); //向摄像头发送水平巡航命令
        $(this).attr("class", "btn btn-info");
        $(this).mouseleave(function() {
            $(this).attr("class", "btn btn-success");
        });
        $(this).mouseup(function() {
            $(this).attr("class", "btn btn-success");
        });
    }
});

$("#ct_v").mousedown(function() { //垂直巡航
    if ((switch_cam == 1) && (camState == 1)) {
        myipcamera.control("VPATROL"); //向摄像头发送垂直巡航命令
        $(this).attr("class", "btn btn-info");
        $(this).mouseleave(function() {
            $(this).attr("class", "btn btn-success");
        });
        $(this).mouseup(function() {
            $(this).attr("class", "btn btn-success");
        });
    }
});

$("#ct_c").mousedown(function() { //360° 巡航

```




```
if ((switch_cam == 1) && (camState == 1)) {  
    myipcamera.control("360PATROL");           //向摄像头发送 360° 巡航命令  
    $(this).attr("class", "btn btn-info");  
    $(this).mouseleave(function() {  
        $(this).attr("class", "btn btn-success");  
    });  
    $(this).mouseup(function() {  
        $(this).attr("class", "btn btn-success");  
    });  
}  
});  
});
```

3.7.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台，1 个网络摄像头，按照任务 3 的方法设置节点板跳线为模式一。

(2) 按照前面所讲内容部署摄像头。

(3) 参考 2.1 节内容部署硬件，组成智云无线传感网络，并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中的智云 ID/KEY、服务器地址、摄像头 IP 地址、摄像头类型、用户名、密码。

(2) 编译 VideoMonitoring 工程，并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入视频监控与采集系统主界面，左侧显示的是一系列控制按钮，右侧是视频监控区域，如图 3.58 所示。

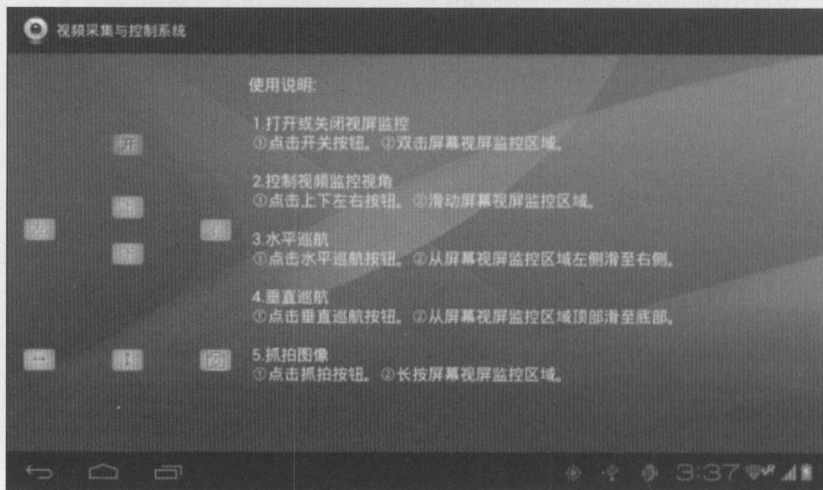


图 3.58 视频监控与采集系统主界面

(4) 单击左侧“开”按钮或者双击右侧视频监控区域屏幕即可打开视频监控，打开摄像头后即可单击“上”、“下”、“左”、“右”按钮或者滑动右侧监控区域来控制摄像头的转动方



向，如图 3.59 所示。



图 3.59 摄像头监控与控制界面

(5) 单击左侧“截屏”按钮或者长按右侧视频监控区域屏幕即可截屏并保存图片，如图 3.60 所示。



图 3.60 保存摄像头当前画面

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中的智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY、摄像头 IP 地址、摄像头类型、用户名、密码。

(2) 将计算机接入互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“VideoMonitoring-Web\VideoMonitoring.html”，进入视频采集与控制网页端实现界面，如图 3.61 所示。



图 3.61 视频采集与控制 Web 端实现

3.7.6 总结与拓展

本任务是基于前面章节知识点的应用及扩展，视频采集与控制系统的开发中，先配置好网络摄像头，然后用实现智云摄像头 API 接口及其相应的方法实现摄像头的控制。

智云物联高级应用开发

本章用 7 个案例，先完成 UI 设计与布局学习，实现的案例有智慧窗帘控制系统开发、自动浇花系统开发、智能门禁系统开发、智能安防系统、实验室管理系统开发、无线抄表系统开发和智能家居自动控制系统开发，通过高级应用项目全面提高开发者的物联网和云平台设计水平。

4.1 任务 19：UI 设计与布局

4.1.1 学习目标

- 掌握 Android 开发之自定义 View 类；
- 掌握 Android 开发之 ActionBar 的使用；
- 掌握 Android 开发之 ViewPager 的使用；
- 掌握 Android 开发之 Fragment 的使用。

4.1.2 开发内容

1. Android 自定义 View

Android 自定义 View 的步骤如下所述。

(1) 自定义 View 的属性：在“res/values/”下建立一个 attrs.xml，在里面定义 View 的属性和声明样式。这里定义了字体、字体颜色、字体大小和图标 4 个属性，format 是该属性的取值类型。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<attr name="icon" format="reference" />
<attr name="color" format="color" />
<attr name="text" format="string" />
<attr name="text_size" format="dimension" />

<declare-styleable name="ChangeColorIconView">
<attr name="icon" />
<attr name="color" />
<attr name="text" />
```



```
<attr name="text_size" />
</declare-styleable>
```

```
</resources>
```

(2) 在 View 的构造方法中，获得自定义的属性。

```
public ChangeColorIconWithTextView(Context context, AttributeSet attrs) {
    super(context, attrs);

    //TypedArray 是一个用来存放由 context.obtainStyledAttributes 获得的属性的数组
    TypedArray a = context.obtainStyledAttributes(attrs,
        R.styleable.ChangeColorIconView);

    int n = a.getIndexCount();
    for (int i = 0; i < n; i++) {

        int attr = a.getIndex(i);
        switch (attr) {
            case R.styleable.ChangeColorIconView_icon:
                BitmapDrawable drawable = (BitmapDrawable) a.getDrawable(attr);
                mIconBitmap = drawable.getBitmap();
                break;
            case R.styleable.ChangeColorIconView_color:
                mColor = a.getColor(attr, 0x45C01A);
                break;
            case R.styleable.ChangeColorIconView_text:
                mText = a.getString(attr);
                break;
            case R.styleable.ChangeColorIconView_text_size:
                mTextSize = (int) a.getDimension(attr, TypedValue
                    .applyDimension(TypedValue.COMPLEX_UNIT_SP, 10,
                        getResources().getDisplayMetrics()));
                break;
        }
    }

    a.recycle();

    mTextPaint = new Paint();
    mTextPaint.setTextSize(mTextSize);
    mTextPaint.setColor(0xff555555);
    //得到 text 绘制范围
    mTextPaint.getTextBounds(mText, 0, mText.length(), mTextBound);
}
```

(3) 重写 View 的 onMeasure 方法。

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);

    //得到绘制 icon 的宽
    int bitmapWidth = Math.min(getMeasuredWidth() - getPaddingLeft()
        - getPaddingRight(), getMeasuredHeight() - getPaddingTop())
}
```



```
- getPaddingBottom() - mTextBound.height());
```

```
int left = getMeasuredWidth() / 2 - bitmapWidth / 2;
int top = (getMeasuredHeight() - mTextBound.height()) / 2 - bitmapWidth / 2;
//设置 icon 的绘制范围
mIconRect = new Rect(left, top, left + bitmapWidth, top + bitmapWidth);
}
```

(4) 重写 View 的 onDraw 方法。

```
@Override
protected void onDraw(Canvas canvas) {
    int alpha = (int) Math.ceil((255 * mAlpha));
    canvas.drawBitmap(mIconBitmap, null, mIconRect, null);
    setupTargetBitmap(alpha);
    drawSourceText(canvas, alpha);
    drawTargetText(canvas, alpha);
    canvas.drawBitmap(mBitmap, 0, 0, null);
}
```

然后就可以在布局中声明的自定义 View 了。使用自定义的属性时需要在当前 xml 文件中声明自己的命名空间，即“xmlns:zonesion=http://schemas.Android.com/apk/res/com.zonesion.autoflowering”。

这样在使用自定义 View 中的属性时引用自己的命名空间即可。

```
<LinearLayout

    Android:layout_width="fill_parent"
    Android:layout_height="60dp"
    Android:background="@drawable/tabbg"
    Android:orientation="horizontal" >

    <com.zonesion.tool.ChangeColorIconWithTextView
        Android:id="@+id/id_indicator_one"
        Android:layout_width="0dp"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
        Android:padding="5dp"
        zonesion:icon="@drawable/chaxun"
        zonesion:text="@string/tab_weixin"
        zonesion:text_size="12sp" />

    <com.zonesion.tool.ChangeColorIconWithTextView
        Android:id="@+id/id_indicator_two"
        Android:layout_width="0dp"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
        Android:padding="5dp"
        zonesion:icon="@drawable/kongzhi"
        zonesion:text="@string/tab_contact"
        zonesion:text_size="12sp" />

    <com.zonesion.tool.ChangeColorIconWithTextView
```




```
        Android:id="@+id/id_indicator_three"
        Android:layout_width="0dp"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
        Android:padding="5dp"
        zonesion:icon="@drawable/guanyu"
        zonesion:text="@string/tab_find"
        zonesion:text_size="12sp" />

<com.zonesion.tool.ChangeColorIconWithTextView
        Android:id="@+id/id_indicator_four"
        Android:layout_width="0dp"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
        Android:padding="5dp"
        zonesion:icon="@drawable/user"
        zonesion:text="@string/tab_me"
        zonesion:text_size="12sp" />
</LinearLayout>
```

2. Android 之 ActionBar 开发

ActionBar 是一种新增的导航栏功能，在 Android 3.0 之后加入系统的 API 中，它标识了读者当前操作界面的位置，并提供了额外的用户动作、界面导航等功能。使用 ActionBar 的好处是，可以提供一种全局统一的 UI 界面，使得开发者在使用任何一款软件时都懂得该如何操作，并且 ActionBar 还可以自动适应各种不同大小的屏幕。

使用 ActionBar 的步骤如下。

(1) 添加 ActionBar: 在 AndroidManifest.xml 中指定 Application 或 Activity 的 theme 是 Theme.Holo 或其子类。

(2) 修改 ActionBar 的图标和标题: 默认情况下，系统会使用<application>或者<activity>中 icon 属性指定的图片来作为 ActionBar 的图标，因此需要修改 ActionBar 的图标可以在<application>或者<activity>中通过 logo 属性来进行指定。而修改标题可以使用 label 属性来指定一个字符串。

(3) 添加 Action 按钮: ActionBar 可以根据应用程序当前的功能来提供与其相关的 Action 按钮，这些按钮都会以图标或文字的形式直接显示在 ActionBar 上。当按钮过多时，多出的一些按钮可以隐藏在 overflow 里面，单击一下 overflow 按钮就可以看到全部的 Action 按钮了。

当 Activity 启动的时候，系统会调用 Activity 的 onCreateOptionsMenu()方法来取出所有的 Action 按钮，只需在这个方法中去加载一个 menu 资源，并把所有的 Action 按钮都定义在资源文件里面就可以了。

```
@Override
//用来显示 ActionBar
public boolean onCreateOptionsMenu(Menu menu) {
    //自定义的 menu
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

定义的 menu 资源文件如下。



```
<menu xmlns:Android="http://schemas.Android.com/apk/res/Android">

<item
    Android:id="@+id/action_search"
    Android:actionViewClass="Android.widget.SearchView"
    Android:icon="@drawable/actionbar_search_icon"
    Android:showAsAction="ifRoom|collapseActionView"
    Android:title="@string/action_search"/>
<item
    Android:id="@+id/action_group_chat"
    Android:icon="@drawable/ofm_group_chat_icon"
    Android:title="@string/action_group_chat"/>
<item
    Android:id="@+id/action_add_friend"
    Android:icon="@drawable/ofm_add_icon"
    Android:title="@string/action_add_friend"/>
<item
    Android:id="@+id/action_scan"
    Android:icon="@drawable/ofm_qrcode_icon"
    Android:title="@string/action_scan"/>
<item
    Android:id="@+id/action_feed"
    Android:icon="@drawable/ofm_feedback_icon"
    Android:title="@string/action_feed"/>

</menu>
```

这里通过 5 个<item>标签定义了 5 个 Action 按钮。<item>标签中又有一些属性，其中 id 是该 Action 按钮的唯一标识符，icon 用于指定该按钮的图标，title 用于指定该按钮可能显示的文字（title 中的内容通常情况下只会在 overflow 中显示出来，ActionBar 中由于屏幕空间有限，默认是不会显示 title 内容的）。其中属性 showAsAction 则指定了该按钮显示的位置，主要有以下几种值可选：always 表示永远显示在 ActionBar 中，如果屏幕空间不够则无法显示，ifRoom 表示屏幕空间够的情况下显示在 ActionBar 中，不够的话就显示在 overflow 中，never 则表示永远显示在 overflow 中。

（4）响应 Action 按钮的单击事件：当开发者单击 Action 按钮的时候，系统会调用 Activity 的 onOptionsItemSelected()方法，通过方法传入的 MenuItem 参数，可以调用它的 getItemId()方法和 menu 资源中的 id 进行比较，从而辨别出开发者单击的是哪一个 Action 按钮。

```
@Override
//ActionBar 中的某个 Action 被单击后会调动此方法
public boolean onOptionsItemSelected(MenuItem item) {
    //TODO Auto-generated method stub
    switch (item.getItemId()) {
        case R.id.action_search:
            Toast.makeText(this, "搜索", Toast.LENGTH_SHORT).show();
            break;
        case R.id.action_scan:
            Toast.makeText(this, "扫一扫", Toast.LENGTH_SHORT).show();
            break;
        case R.id.action_add_friend:
```



```

        Toast.makeText(this, "添加", Toast.LENGTH_SHORT).show();
        break;
        case R.id.action_group_chat:
            Toast.makeText(this, "群聊", Toast.LENGTH_SHORT).show();
            break;
        case R.id.action_feed:
            Toast.makeText(this, "意见反馈", Toast.LENGTH_SHORT).show();
            break;
        case Android.R.id.home:
            Toast.makeText(this, "返回至主界面", Toast.LENGTH_SHORT).show();
            resetOtherTabs(0);
            mFragmentIndicator.get(0).setIconAlpha(1.0f);
            mViewPager.setCurrentItem(0, false);
            break;
    }
    Return true;
}

```

(5) 显示 overflow 按钮: overflow 按钮的显示情况和设备的硬件情况有关, 如果设备没有物理 Menu 键的话, overflow 按钮就可以显示, 如果有物理 menu 键的话, overflow 按钮就不会显示出来。为了使得程序在不同设备上都可以显示 overflow 按钮, 需要将 ViewConfiguration 这个类中的静态变量 sHasPermanentMenuKey 的值设为 false。

```

//使用反射的方式将 sHasPermanentMenuKey 的值设置成 false, 即使是在有 menu 键的手机上,
//也能让 overflow 按钮显示出来
private void setOverflowShowingAlways() {
    try {
        //设备是否有物理 menu 键
        //true if a permanent menu key is present, false otherwise.
        ViewConfiguration config = ViewConfiguration.get(this);
        Field menuKeyField = ViewConfiguration.class
            .getDeclaredField("sHasPermanentMenuKey");
        menuKeyField.setAccessible(true);
        menuKeyField.setBoolean(config, false);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

(6) 让 overflow 中的按钮显示对应的图标: Google 官方默认隐藏在 overflow 中的 Action 按钮都应该只显示文字。为了使 overflow 中的 Action 按钮也可以显示图标, 需要将 MenuBuilder 这个类的 setOptionalIconsVisible 变量值设为 true。

```

@Override
/*
 * overflow 被展开时就会回调这个方法,
 * 通过返回反射的方法将 MenuBuilder 的 setOptionalIconsVisible 变量设置为 true,
 * 使得 overflow 中的 Action 按钮对应的图标显示出来
 */
public boolean onMenuOpened(int featureId, Menu menu) {
    if (featureId == Window.FEATURE_ACTION_BAR && menu != null) {
        if (menu.getClass().getSimpleName().equals("MenuBuilder")) {

```




```

try {
    Method m = menu.getClass().getDeclaredMethod(
        "setOptionalIconsVisible", Boolean.TYPE);
    m.setAccessible(true);
    m.invoke(menu, true);
} catch (Exception e) {
}
}
}
Return super.onMenuOpened(featureId, menu);
}

```

3. Android 之 ViewPager 使用

简单地说, ViewPager 的功能就是使视图左右滑动, ViewPager 组件只用来显示左右滑动的界面, 使用 ViewPager 的步骤如下。

(1) 在布局文件中声明 ViewPager。

```

<Android.support.v4.view.ViewPager
    Android:id="@+id/id_viewpager"
    Android:layout_width="fill_parent"
    Android:layout_height="0dp"
    Android:layout_weight="1">
</Android.support.v4.view.ViewPager>

```

(2) 在 Activity 里实例化 ViewPager 组件并设置适配器 Adapter 及其数据源。由于 ViewPager 适配器默认的是预加载前后两个页面, 因此为了避免 fragment 切换过程中视图重建, 这里调用方法 “mViewPager.setOffscreenPageLimit(3)” 将预加载数量修改为 (fragment 数量-1) 即可。

```

//声明并初始化 Fragment 数组
private List<Fragment> mFragments = new ArrayList<Fragment>();
private FragmentPagerAdapter mAdapter; //声明 ViewPager 的适配器
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mViewPager = (ViewPager) findViewById(R.id.id_viewpager); //获取 ViewPager
控件
    mViewPager.setAdapter(mAdapter); //为 mViewPager 设置适配器
    mViewPager.setOffscreenPageLimit(3); //设置缓冲数
    initDatas();
}
private void initDatas() {
    //实例化 Fragment
    FirstFragment firstFragment = new FirstFragment();
    SecondFragment secondFragment = new SecondFragment();
    ThirdFragment thirdFragment = new ThirdFragment();
    FourthFragment fourthFragment = new FourthFragment();
    //设置数据源
    mFragments.add(firstFragment);
    mFragments.add(secondFragment);
    mFragments.add(thirdFragment);
    mFragments.add(fourthFragment);
    mAdapter = new FragmentPagerAdapter(getSupportFragmentManager()) {

```



```

        @Override
        public int getCount() {
            return mFragments.size();
        }
        @Override
        public Fragment getItem(int arg0) {
            return mFragments.get(arg0);
        }
    };
}

```

(3) 为 ViewPager 设置监听。

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    //为 mViewPager 设置页面滑动监听器，让 Activity 去实现监听
    mViewPager.setOnPageChangeListener(this);
}
@Override
//页面跳转完后得到调用，arg0 是当前页面的 position
public void onPageSelected(int arg0) {
    mFragmentIndicator.get(arg0).setIconAlpha(1.0f);
    //其他的 tab 的透明度就设置为 0
    resetOtherTabs(arg0);
}
@Override
/*
*参数说明: arg0 :当前页面; arg1:当前页面偏移的百分比; arg2:当前页面偏移的像素位置
*方法的作用: 当页面在滑动的时候会调用此方法，在滑动被停止之前，此方法会一直得到调用
*/
public void onPageScrolled(int position, float positionOffset,
                           int positionOffsetPixels) {
}
@Override
public void onPageScrollStateChanged(int state) {
}

```

4. Android 之 Fragment 使用

Fragment 拥有自己的生命周期和接收、处理用户的事件，这样就不必在 Activity 写一堆控件的事件处理的代码了。Fragment 可以当成 Activity 的一个界面的一个组成部分，甚至 Activity 的界面可以完全由不同的 Fragment 组成，可以在 Activity 中动态的添加、替换和移除某个 Fragment。

由于本任务中使用的 Fragment 并不是很多，因此使用的是静态加载 Fragment 的方法。静态加载 Fragment 的方法很简单。

(1) 继承 Fragment，重写 onCreateView 决定 Fragment 的布局。

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    //生成 fragment 视图的布局
    View v = inflater.inflate(R.layout.first_fragment, container, false);
    //将生成的视图返回给托管 Activity
}

```



```
return v;  
}
```

(2) 在 Activity 中声明并实例化 Fragment 即可。

```
//实例化 Fragment  
FirstFragment firstFragment = new FirstFragment();  
SecondFragment secondFragment = new SecondFragment();  
ThirdFragment thirdFragment = new ThirdFragment();  
FourthFragment fourthFragment = new FourthFragment();
```

4.1.3 开发步骤

编译 SimpleUI 工程，并安装应用程序到 S210 系列 Android 开发平台，程序运行后的主界面如图 4.1 所示。

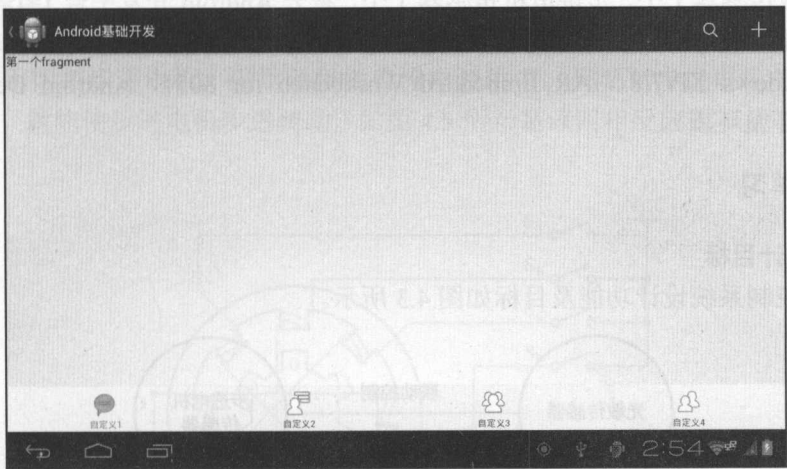


图 4.1 程序主界面

开发者可以单击自定义的 View 或者左右滑动界面来切换页面；单击右上角的“+”图标，会弹出 ActionBar 控件中未能在屏幕上方显示的 Action，如图 4.2 所示。

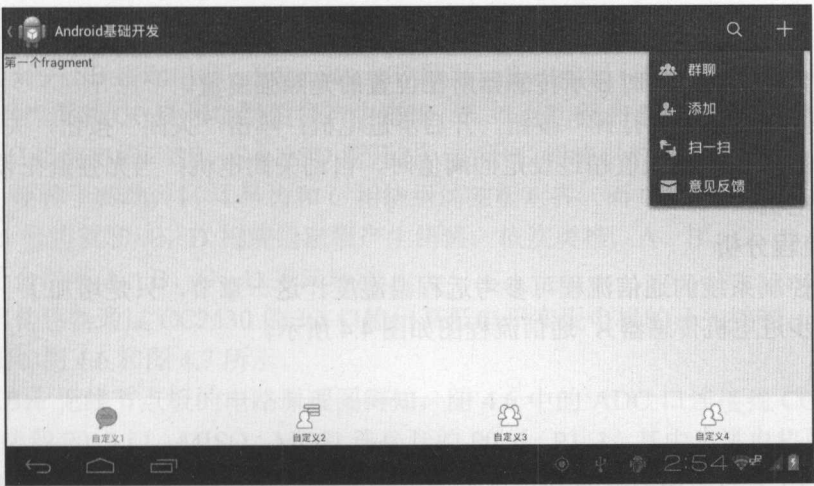


图 4.2 ActionBar 控件显示



4.2 任务 20：智慧窗帘控制系统开发（案例 8）

4.2.1 学习目标

- 掌握 Android 开发之 ViewPager 的使用；
- 掌握 Android UI 开发之 RadioGroup 的使用；
- 学会智慧窗控制系统项目开发和调试。

4.2.2 开发环境

硬件：光敏传感器 1 个，步进电机传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

4.2.3 原理学习

1. 系统设计目标

智慧窗帘控制系统设计功能及目标如图 4.3 所示。

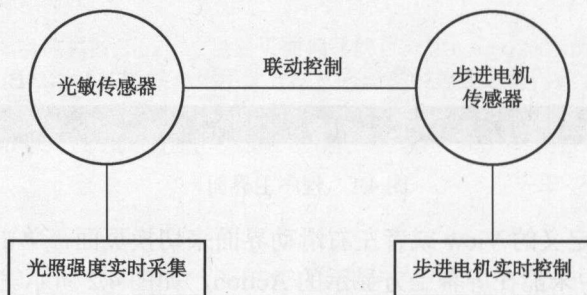


图 4.3 智慧窗帘系统功能模块

- (1) 实时数据采集：实时显示传感器所在位置的光照强度值。
- (2) 执行控制：单击“打开”按钮，开启步进电机；单击“关闭”按钮，关闭步进电机。
- (3) 联动控制：当光强值超过设定的阈值时，自动关闭电机；当光强值在设定的阈值内时，自动开启电机。

2. 业务流程分析

智慧窗帘控制系统的通信流程可参考远程温湿度计这一章节，只是增加了一个控制类的传感器节点（步进电机传感器），通信流程图如图 4.4 所示。

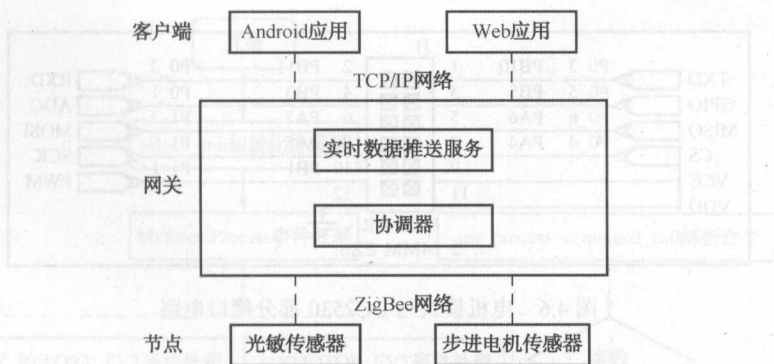


图 4.4 智慧窗帘系统通信流程

3. 硬件原理

光敏传感器的硬件原理可参考农作物光强监测这一章节的相关内容。

本任务使用四相步进电机，采用单极性直流电源供电。只要对步进电机的各相绕组按合适的时序起电，就能使步进电机步进转动，如图 4.5 所示是该四相反应式步进电机工作原理示意图。

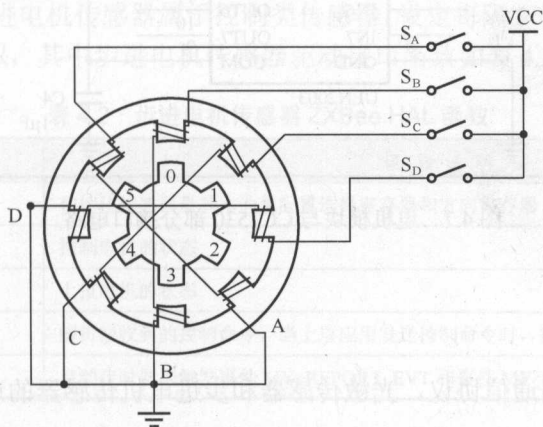


图 4.5 四相步进电机步进示意图

开始时，开关 SB 接起电源，SA、SC、SD 断开，B 相磁极和转子 0、3 号齿对齐，同时，转子的 1、4 号齿就和 C、D 相绕组磁极产生错齿，2、5 号齿就和 D、A 相绕组磁极产生错齿。

当开关 SC 接起电源，SB、SA、SD 断开时，由于 C 相绕组的磁力线和 1、4 号齿之间磁力线的作用，使转子转动，1、4 号齿和 C 相绕组的磁极对齐。而 0、3 号齿和 A、B 相绕组产生错齿，2、5 号齿就和 A、D 相绕组磁极产生错齿。依次类推，A、B、C、D 四相绕组轮流供电，则转子会沿着 A、B、C、D 方向转动。

步进电机传感器通过 CC2530 的 IO 口输出高低电平实现电机的开关控制，与 CC2530 部分接口电路图如图 4.6 和图 4.7 所示。

根据 ZXBee 无线节点板的电路原理图得知，图 4.6 中的 ADC 口连接到 CC2530 的 P0_1 口，GPIO 连接到 P0_5 口，MISO、MOSI 连接到的 P0_6、P1_3。其中步进电机的 A 相绕组连接到 P0_5，B 相绕组连接到 P0_1，C 相绕组连接到 P0_6，D 相绕组连接到 P1_3，要让步进电机工作起来，只要改变 A、B、C、D 的电平变化次序即可。

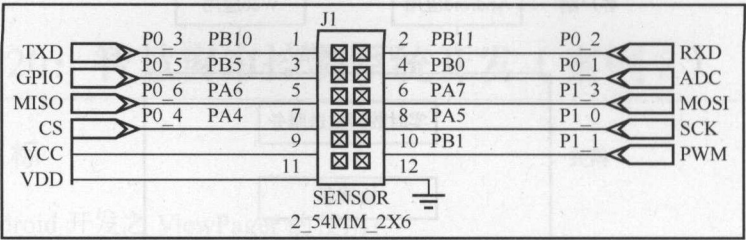


图 4.6 电机模块与 CC2530 部分接口电路

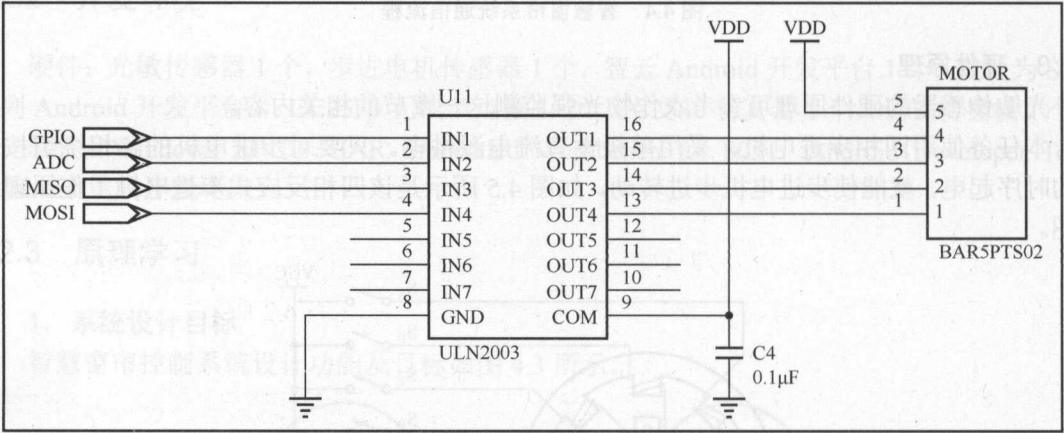


图 4.7 电机模块与 CC2530 部分接口电路

4.2.4 开发内容

1. 硬件层驱动设计

(1) ZXBee 智云数据通信协议。光敏传感器和步进电机传感器的通信协议如表 4.1 所示。

表 4.1 相关传感器智云通信协议定义

传 感 器	属 性	参 数	权 限	说 明
光敏传感器	数值	A0	R	光强值，浮点型：0.1 精度
步进电机	电机开关	D1(OD1/CD1)	R(W)	0 或者 1 变化

(2) 传感器驱动程序开发。光敏传感器的驱动程序开发可参考农作物光强监测这一章节的相关内容。步进电机传感器的驱动开发设计如图 4.8 所示。

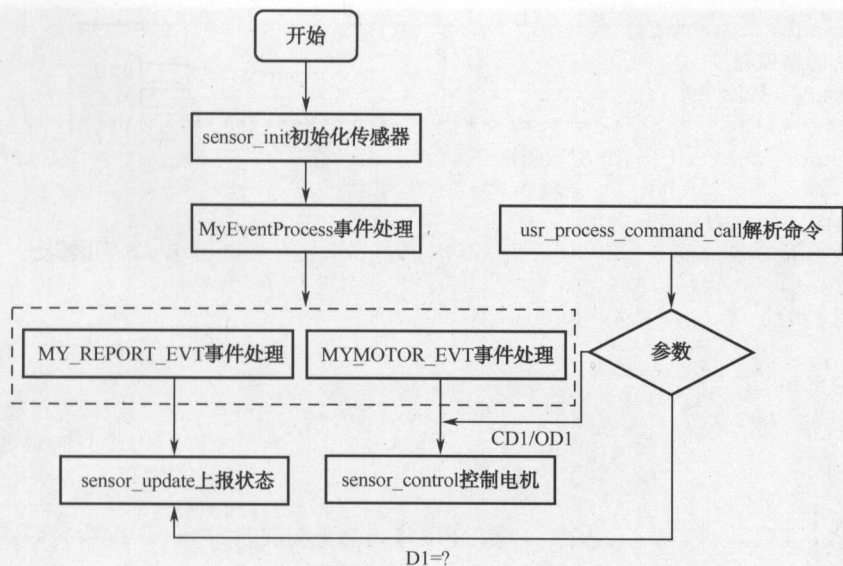


图 4.8 步进电机传感器程序逻辑

根据 2.1 节所述，步进电机传感器属于控制类传感器，设定每隔 30 s 主动上报传感器数值，与继电器的工作原理类似，其中步进电机传感器驱动接口函数如表 4.2 所示。

表 4.2 步进电机传感器 ZXBee HAL 函数

核心函数名	函 数 说 明
sensor_init()	初始化传感器最基本的是配置选择寄存器和方向寄存器
sensor_control()	控制电机的状态
sensor_update()	上报电机的状态
usr_process_command_call()	解析接收到的控制命令，当上层应用发送控制命令时，指定该函数进行命令解析
MyEventProcess()	启动定时器来触发事件 MY_REPORT_EVT 和事件 MY_DCMOTOR_EVT

部分程序代码如下。

```

/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    motor_init();

    //启动定时器，触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10)
                                                                *1000));

    osal_start_timerEx( sapi_TaskID, MY_MOTOR_EVT, 1);
}

/*****
```



```
*名称: sensor_control()
*功能: 传感器控制
*参数: cmd - 控制命令
*****/
void sensor_control(uint8 cmd)
{
    if((D1 & 0x01) == 0){
        Flag = 2;                                //停止转动
    } else {
        if((D1 & 0x02) == 0x02){
            Flag = 0;                                //反转
        } else {
            Flag = 1;                                //正转
        }
    }
}
/*****
*名称: updateV0()
*功能: 更新V0的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的V0值
*****/
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}
/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){ //若控制编码上报允许, 则 pData 的数据包中添加控制编码数据
        len = sprintf((char*)p, "D1=%u", D1);
        p += len;
        *p++ = ',';
    }

    //将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
```



```

if (p - pData > 1) {
    pData[0] = '{';
    p[0] = 0;
    p[-1] = '}';

    zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                                                                AF_DEFAULT_RADIUS );
    HalledSet( HAL_LED_1, HAL_LED_MODE_BLINK );    //通信 LED 闪烁一次
}
}

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest{}发送给协
           调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);
        }
    }
    if (0 == strcmp("CD1", ptag)) {
        D1 &= ~val;
        sensor_control(D1);
    }
    if (0 == strcmp("OD1", ptag)) {
        D1 |= val;
        sensor_control(D1);
    }
    if (0 == strcmp("D1", ptag)) {
        if (0 == strcmp("?", pval)) {

```




```
        ret = sprintf(pout, "D1=%u", D1);
    }
}
if (0 == strcmp("V0", ptag)) {
    if (0 == strcmp("?", pval)) {
        ret = sprintf(pout, "V0=%u", V0);
    }else{
        updateV0(pval);
    }
}

return ret;
}

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件 MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
                                                                *1000));
    }

    //循环处理电机转动状态
    if ( event & MY_MOTOR_EVT )
    {
        if(Flag == 2){                                     //停止
            motor_stop();
        }
        if(Flag == 1){
            motor_step(1);
        }
        if(Flag == 0){                                     //反转
            motor_step(0);
        }
        osal_start_timerEx( sapi_TaskID, MY_MOTOR_EVT, 1);
    }
}
```

2. 移动端应用设计

(1) 工程框架介绍。智慧窗帘系统工程框架如表 4.3 所示。



表 4.3 智慧窗帘系统工程框架

包名（类名）	说 明
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象，定义应用程序全局单例对象
com.zonesion.view 工具包	
ChangeColorIconWithTextView	自定义 View 的实现
com.zonesion.fragment 子模块包	
SmartCurtainFragment.java	光强值实时查询和步进电机控制模块
TabFragment.java	供开发者自定义模块
com.zonesion.activity activity 包	
MainActivity.java	主 Activity

(2) 程序业务流程分析。智慧窗帘控制系统调用的是实时数据 API 接口，相比第 2 章，只是将采集类传感器和控制类传感器结合在一起，程序的实现流程如图 4.9 所示。

(3) 程序代码剖析。

① ZApplication 框架说明参考智能灯光控制系统这一章节的相关内容。

② MainActivity 实现了 ViewPager 数据源设置，各个 Fragment 模块加载，以及自定义 View 的单击事件和 ActionBar 控件的应用，核心源码如下所示。

```
public class MainActivity extends FragmentActivity implements
    OnPageChangeListener, OnClickListener {
    .....
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        .....
        setOverflowShowingAlways(); //显示 ActionBar 导航栏的 Overflow 按钮
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true); //使用程序图标作为 home icon
        actionBar.setDisplayHomeAsUpEnabled(true); //显示返回的箭头，并可通过
            //onOptionsItemSelected() 进行监听，其资源 ID 为 Android.R.id.home。
        //获取 ViewPager 控件
        mViewPager = (ViewPager) findViewById(R.id.id_viewpager);
        //实例化 Fragment 及 Fragment 指示器
        initDatas();
        mViewPager.setAdapter(mAdapter); //为 mViewPager 设置适配器
        //为 mViewPager 设置页面滑动监听器，让 Activity 去实现监听}
        mViewPager.setOnPageChangeListener(this);
        .....
    }
}
```

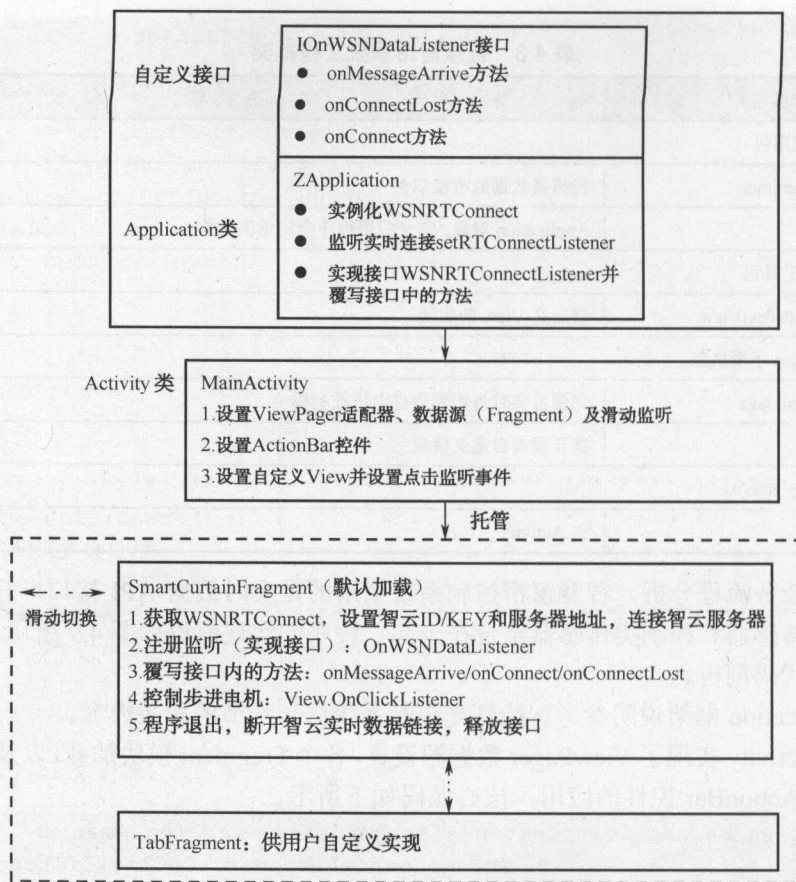


图 4.9 窗帘控制系统程序实现流程

③ 光强值实时数据采集和电机控制(SoilFragment)。通过 `mApplication.getWSNRConnect()` 获取 `WSNRConnect` 实例，设置 ID/KEY 和服务器地址，通过 `mApplication.registerOnWSNDataListener (this)` 注册传感器数据监听（实现接口 `IONWSNDataListener`），建立实时连接。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mApplication = (ZApplication) getActivity().getApplication();
    wRTConnect = mApplication.getWSNRConnect();
    wRTConnect.setIdKey(ID, KEY); //设置 ID/KEY
    wRTConnect.setServerAddr("zhiyun360.com:28081"); //设置服务器地址
    mApplication.registerOnWSNDataListener (SoilFragment.this); //注册监听
    wRTConnect.connect(); //建立实时数据服务连接
}
```

覆写接口的方法：在 `onConnect()` 方法中发送查询光强值和电机状态的命令，即在连接成功后立即查询光强值而不是一直等待底层主动上传数据。

```
@Override
public void onConnect() { //连接成功时发送查询命令
```




```
//TODO Auto-generated method stub
mWSNRTConnect.sendMessage(curtain_mac, "{D1=?}".getBytes());
mWSNRTConnect.sendMessage(light_mac, "{A0=?}".getBytes());
}
```

在 `onMessageArrive()` 方法中解析获取到的传感器数据：将光强值显示在视图中。

```
@Override
public void onMessageArrive(String mac, String tag, String val) {
    if (light_mac.equalsIgnoreCase(mac)) {
        if (tag.equals("A0")) {
            float fValue = Float.parseFloat(val);
            tvLightIntensity.setText("当前光照强度为: " + fValue); // 设置文本
            if (automaticFlag) {
                if (fValue > automaticValue) {
                    Log.d("Automatic", "fValue=" + fValue + ", auto="
                        + automaticValue);
                    mWSNRTConnect.sendMessage(curtain_mac, "{CD1=1,D1=?}"
                        .getBytes());
                } else {
                    mWSNRTConnect.sendMessage(curtain_mac,
                        "{OD1=1,D1=?}".getBytes());
                }
            }
        }
    }
    if (curtain_mac.equalsIgnoreCase(mac)) {
        if (tag.equals("D1")) {
            int v = Integer.parseInt(val);
            if ((v & 0x01) == 0x01) {
                ivSwitch.setImageResource(R.drawable.curtain_switch_on);
                bSwitchFlag = true;
                ivCurtain.setImageResource(R.drawable.curtain_state_on);
            } else {
                ivSwitch.setImageResource(R.drawable.curtain_switch_off);
                bSwitchFlag = false;
                ivCurtain.setImageResource(R.drawable.curtain_state_off);
            }
        }
    }
}
```

单击按钮发送 “{OD1=1, D1=?}{CD1=1, D1=?}” 命令来控制电机的开关。

//单击监听事件

```
View.OnClickListener mOnClickListener = new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        if (bSwitchFlag == false) {
            mWSNRTConnect.sendMessage(curtain_mac, "{OD1=1,D1=?}".getBytes());
            Toast.makeText(getActivity(), "{OD1=1,D1=?}",
                Toast.LENGTH_SHORT).show();
        } elseif (bSwitchFlag == true) {
```



```

        mWSNRTConnect.sendMessage(curtain_mac, "{CD1=1,D1=?}".getBytes());
        Toast.makeText(getActivity(), "{CD1=1,D1=?}",
                                Toast.LENGTH_SHORT).show();
    }
    mWSNRTConnect.sendMessage(light_mac, "{A0=?}".getBytes());
}
};

```

④ RadioGroup 使用。在 XML 文件中定义 RadioGroup 控件。

```

<RadioGroup
    Android:id="@+id/rg_control"
    Android:layout_width="match_parent"
    Android:layout_height="wrap_content"
    Android:gravity="center"
    Android:orientation="vertical">
    <RadioButton
        Android:id="@+id/rb_automatic"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content"
        Android:text="@string/automatic" />
    <RadioButton
        Android:id="@+id/rb_manual"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content"
        Android:text="@string/manual" />
</RadioGroup>

```

在 Java 文件中获取 RadioGroup 控件并设置监听器，覆写其单击事件的方法。

```

rgControl = (RadioGroup) view.findViewById(R.id.rg_control);
rbAutomatic = (RadioButton) view.findViewById(R.id.rb_automatic);
rbManual = (RadioButton) view.findViewById(R.id.rb_manual);
//单选按钮监听事件
RadioGroup.OnCheckedChangeListener mOnCheckedChangeListener = new
    RadioGroup.OnCheckedChangeListener() {

    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        if (checkedId == rbAutomatic.getId()) {
            mWSNRTConnect.sendMessage(light_mac, "{A0=?}".getBytes());
            automaticFlag = true;
            tvAutomaticValue.setVisibility(View.VISIBLE);
            sbAutomaticValue.setVisibility(View.VISIBLE);
            ivSwitch.setVisibility(View.INVISIBLE);
        } else if (checkedId == rbManual.getId()) {
            mWSNRTConnect.sendMessage(curtain_mac, "{D1=?}".getBytes());
            automaticFlag = false;
            tvAutomaticValue.setVisibility(View.INVISIBLE);
            sbAutomaticValue.setVisibility(View.INVISIBLE);
            ivSwitch.setVisibility(View.VISIBLE);
        }
    }
};

```



3. Web 端应用设计

根据 Web 应用编程接口定义,智慧窗帘控制系统的应用设计主要采用实时数据 API 接口,JS 部分代码如下。

```
var rtc = new WSNRTConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
rtc.connect(); //数据推送服务连接
$("#ConnectState").text("数据服务连接中...");
rtc.onConnect = function () { //连接成功回调函数
    rtc.sendMessage(mySensorMac1, "{A0=?}"); //向光敏传感器发送数据
    //向步进电机发送数据(查询电机状态)
    rtc.sendMessage(mySensorMac2, "{D1=?,CD0=1}");
    $("#ConnectState").text("数据服务连接成功!");
};
rtc.onConnectLost = function () { //数据服务掉线回调函数
    $("#ConnectState").text("数据服务掉线!");
};
rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
    console.log(mac, " >>> ", dat);
    if (mac != mySensorMac1 && mac != mySensorMac2){ //判断传感器 MAC 地址
        console.log("'" + mac + " not in sensors");
        return;
    }
    if (dat[0] == '{' && dat[dat.length - 1] == '}'){ //判断字符串首尾是否为{}
        dat = dat.substr(1, dat.length - 2); //截取{}内的字符串
        var its = dat.split(','); //以','来分割字符串
        for (var x in its) {
            var t = its[x].split('='); //以'='来分割字符串
            if (t.length != 2) continue;
            if (t[0] == "A0") { //判断参数 A0
                $("#LightIntensity").text(t[1]); //显示接收到的原始数据
                lightIntensity = t[1];
            }
            if (t[0] == "D1") { //判断参数 D1
                var anNiu = document.getElementById("button");
                var bG = document.getElementById("bg");
                if (t[1] == "0" || t[1] == "1") { //判断步进电机状态
                    anNiu.src = ("images/cl-an-on.png");
                    bG.src = ("images/cl-on.gif");
                    CurtainState = true; //窗帘状态为开
                }
                if (t[1] == "2" || t[1] == "3") { //判断步进电机状态
                    anNiu.src = ("images/cl-an-off.png");
                    bG.src = ("images/cl-off.gif");
                    CurtainState = false; //窗帘状态为关
                }
            }
        }
    }
};
```




```

$(function () {
    elem01 = document.querySelector('.js-min-max-start');//选择 input 元素
    init01 = new Powerange(elem01, { min: 0, max: 700, start: 300,
        callback:function(){ //实例化 powerange 类并且初始化参数
            $("#range").text(elem01.value); //显示当前阈值

            var IsChecked = document.getElementById("checkboxoid").checked;
            if (IsChecked == true) { //若复选框被选中
                if (parseInt(lightIntensity) < parseInt(elem01.value) &&
                    CurtainState==false) { //如果当前光照强度低于阈值且窗帘状态
                    document.getElementById("bg").src = ("images/cl-on.gif");
                    document.getElementById("button").src =
                        ("images/cl-an-on.png");

                    //向步进电机发送数据（正转）
                    rtc.sendMessage(mySensorMac2, "{OD1=1,CD1=2}");
                    setTimeout( 'rtc.sendMessage(mySensorMac2,"{CD1=3}"',10000);
                    CurtainState = true;
                }
            }
        }
    });
    $("#range").text(elem01.value);
});

function anniu() {
    var IsChecked = document.getElementById("radioid").checked;
    if (IsChecked == true) {
        if (CurtainState) {
            //向步进电机发送数据（反转）
            rtc.sendMessage(mySensorMac2, "{OD1=3,D1=?}");
            setTimeout('rtc.sendMessage(mySensorMac2, "{CD1=1,OD1=2}"',10000);
        }
        else {
            //向步进电机发送数据（正转）
            rtc.sendMessage(mySensorMac2, "{OD1=1,CD1=2,D1=?}");
            setTimeout( 'rtc.sendMessage(mySensorMac2, "{CD1=3}"',10000);
        }
    }
}

function AutoControl(){
    var IsChecked = document.getElementById("checkboxoid").checked;
    if (IsChecked == true) {
        if (parseInt(lightIntensity) < parseInt(elem01.value) &&
            CurtainState==false) { //如果当前光照强度低于阈值
            //向步进电机发送数据（正转）
            rtc.sendMessage(mySensorMac2, "{OD1=1,CD1=2,D1=?}");
            setTimeout( 'rtc.sendMessage(mySensorMac2, "{CD1=3}"',10000);
        }
    }
}

```



4.2.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个光敏传感器, 一个步进电机传感器, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到 “C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples” 文件夹下。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 SmartCurtain 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入智慧窗帘系统主界面, 在主界面弹出“连接网关成功”消息后即表示连接到智云服务中心。

(4) 连接网关成功后会发送查询光强值的命令并将光强值在左侧显示出来, 开发者可以选择自动控制或者手动控制模式来控制电机的开关, 系统默认选中手动模式, 如图 4.10 所示。

(5) 选择自动控制, 系统默认的初始阈值为 200, 开发者也可以调节光强阈值, 如图 4.11 所示。

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址 (若在局域网内使用, 则设置为智云 Android 开发平台的 IP) 和智云 ID/KEY。

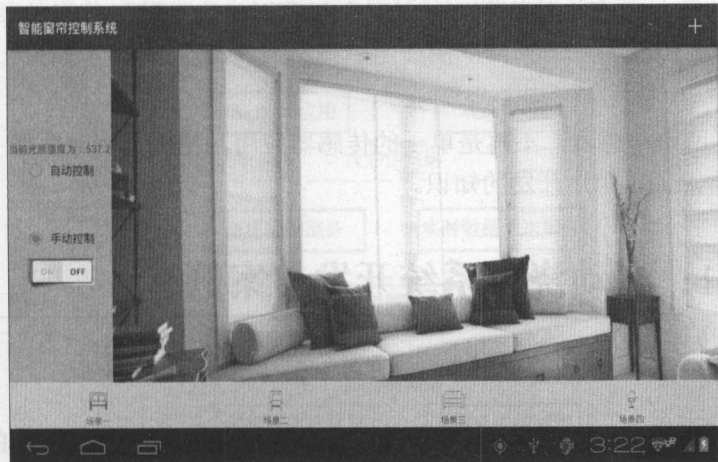


图 4.10 光强值实时显示和电机控制



图 4.11 窗帘控制系统的自动控制模式

(2) 电脑接入互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“SmartCurtain-Web\SmartCurtain.html”，进入智慧窗帘控制系统界面，在主界面右上角显示“数据服务连接成功！”消息后即表示连接到智云服务中心，在左侧栏会实时地显示当前光强值，在左侧可选择自动控制模式并设置光强阈值，也可以选择手动控制模式来控制电机的开关，如图 4.12 所示。

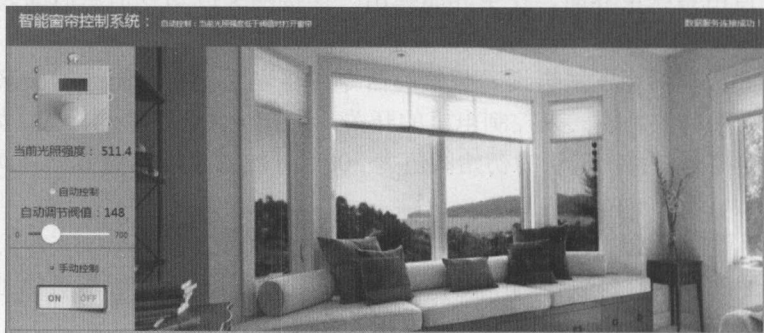


图 4.12 智慧窗帘控制系统网页端设计效果图

4.2.6 总结与拓展

本任务用到了两个传感器，不再是单一的传感器应用，使用了实时数据 API 接口，另外还学习了一些关于 Android UI 开发的知识。

4.3 任务 21：自动浇花系统开发（案例 9）

4.3.1 学习目标

- 掌握智云通信协议 sensor_control()函数和 MY_DCMOTOR_EVT 事件的运用；
- 掌握 Android 开发之 ViewPager 的使用；
- 掌握 Android UI 开发之 SeekBar、CheckBox 和 Spinner 的使用。



4.3.2 开发环境

硬件：土壤温湿度传感器 1 个，直流电机传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

4.3.3 原理学习

1. 系统设计目标

自动浇花系统设计功能及目标如图 4.13 所示。

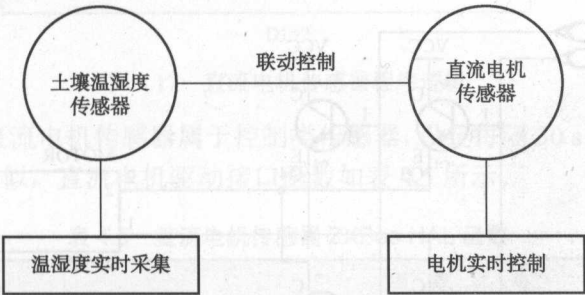


图 4.13 自动浇花系统功能模块

- (1) 实时数据采集：实时显示传感器所在位置的土壤温湿度的值。
- (2) 执行控制：单击“打开”按钮，开启电机；单击“关闭”按钮，关闭电机。
- (3) 联动控制：当土壤温湿度值超过设定的阈值时，自动开启电机；当温湿度值在设定的阈值内时，自动关闭电机。

2. 业务流程分析

自动浇花系统的通信流程可参考远程温湿度计这一章节相关内容，只是增加了一个控制类的传感器节点（直流电机传感器），通信流程图如图 4.14 所示。

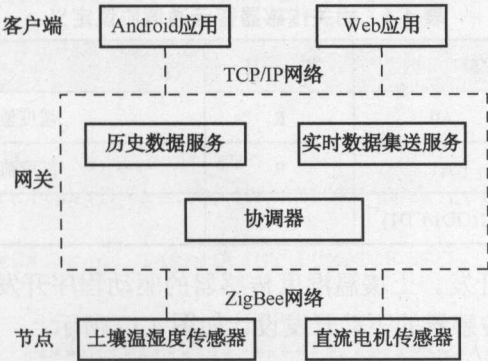


图 4.14 自动浇花系统通信流程



3. 硬件原理

土壤温湿度传感器的硬件原理参考远程温湿度计这一章节的相关内容。

直流电机传感器通过 CC2530 的 IO 口输出高低电平实现直流电机的开关控制,与 CC2530 部分接口电路图如图 4.15 和图 4.16 所示。

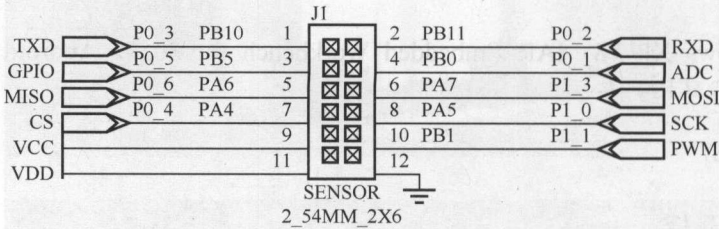


图 4.15 电机模块与 CC2530 部分接口电路

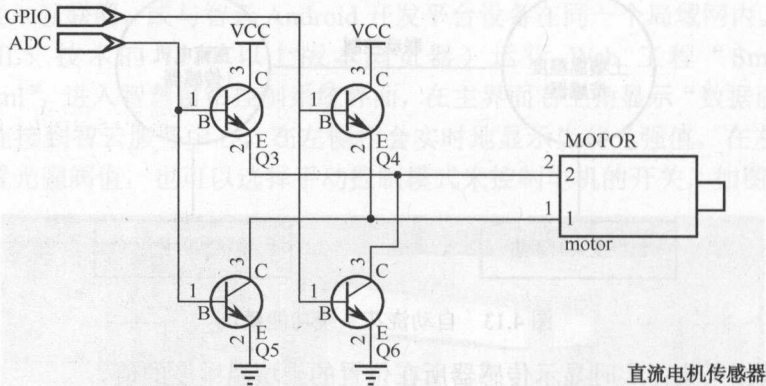


图 4.16 电机模块与 CC2530 部分接口电路

4.3.4 开发内容

1. 硬件层驱动设计

(1)ZXBee 智云数据通信协议。土壤温湿度传感器和电机传感器的通信协议如表 4.4 所示。

表 4.4 相关传感器智云通信协议定义

传感器	属 性	参 数	权 限	说 明
土壤温湿度	温度值	A0	R	温度值, 浮点型: 0.1 精度
	湿度值	A1	R	湿度值, 浮点型: 0.1 精度
直流电机	电机开关	D1(OD1/CD1)	R(W)	0 或者 1 变化

(2) 传感器驱动程序开发。土壤温湿度传感器的驱动程序开发参考远程温湿度计这一章节的相关内容, 直流电机传感器的驱动开发设计如图 4.17 所示。

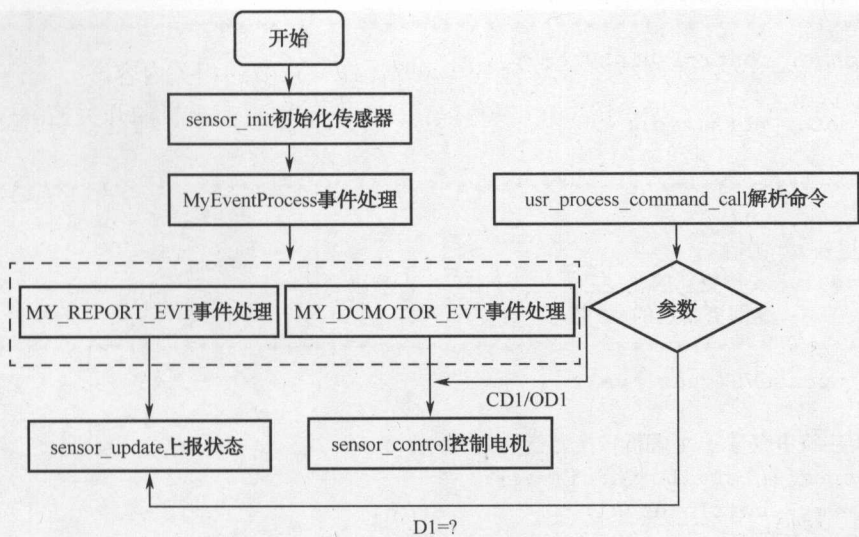


图 4.17 直流电机传感器程序逻辑

根据 2.1 节所述，直流电机传感器属于控制类传感器，设定每隔 30 s 主动上报传感器数值，与继电器的工作原理类似，直流电机驱动接口函数如表 4.5 所示。

表 4.5 直流电机传感器 ZXBee HAL 函数

核 心 函 数	函 数 说 明
sensor_init()	初始化传感器最基本的是配置选择寄存器和方向寄存器
sensor_control()	控制电机的状态
sensor_update()	上报电机的状态
usr_process_command_call()	解析接收到的控制命令，当上层应用发送控制命令时，指定该函数进行命令解析
MyEventProcess()	启动定时器来触发事件 MY_REPORT_EVT 和事件 MY_DCMOTOR_EVT

部分程序代码如下。

```

/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    dc_motor_init();
    //启动定时器，触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10)
                                                                *1000));
    osal_start_timerEx( sapi_TaskID, MY_DCMOTOR_EVT, 1);
}

/*****
*名称: sensor_control()
*功能: 传感器控制
*参数: cmd -- 控制命令
*****/
```




```
*****/
void sensor_control(uint8 cmd)
{
    dc_motor_work(cmd);
}
/*****
*名称: updateV0()
*功能: 更新 V0 的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的 V0 值
*****/
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;
    return V0;
}

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest{} 发送给协
           调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);
        }
    }
    if (0 == strcmp("CD1", ptag)) {
        D1 &= ~val;
    }
}
```

```
        sensor_control(D1);
    }
    if (0 == strcmp("OD1", ptag)) {
        D1 |= val;
        sensor_control(D1);
    }
    if (0 == strcmp("D1", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D1=%u", D1);
        }
    }
    if (0 == strcmp("V0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "V0=%u", V0);
        }else{
            updateV0(pval);
        }
    }
}
return ret;
}

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器，触发事件: MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
                                                                    *1000));
    }

    //循环处理电机转动状态
    if ( event & MY_DCMOTOR_EVT )
    {
        sensor_control(D1);
        osal_start_timerEx( sapi_TaskID, MY_DCMOTOR_EVT, 1);
    }
}
```

2. 移动端应用设计

(1) 工程框架介绍。自动浇花系统工程框架如表 4.6 所示。

表 4.6 自动浇花系统工程框架

包名（类名）	说 明
com.zonesion.app 应用包	
IOOnWSNDataListener.java	传感器数据监听接口类



续表

包名（类名）	说 明
ZApplication.java	Application 对象，定义应用程序全局单例对象
com.zonesion.tool 工具包	
ChangeColorIconWithTextView	自定义 View 的实现
MyDialog	自定义 ProgressDialog 的实现
com.zonesion.ui 子模块包	
SoilFragment.java	土壤温湿度实时查询和电机控制模块
HistoryChartFragment.java	历史数据查询模块
SettingFragment.java	自动控制设置模块
AboutFragment.java	关于模块
com.zonesion.activity activity 包	
MainActivity.java	主 Activity

(2) 程序业务流程分析。自动浇花系统调用的是实时数据 API 接口和历史数据 API 接口，将采集类传感器和控制类传感器结合在一起，程序的实现流程如图 4.18 所示。

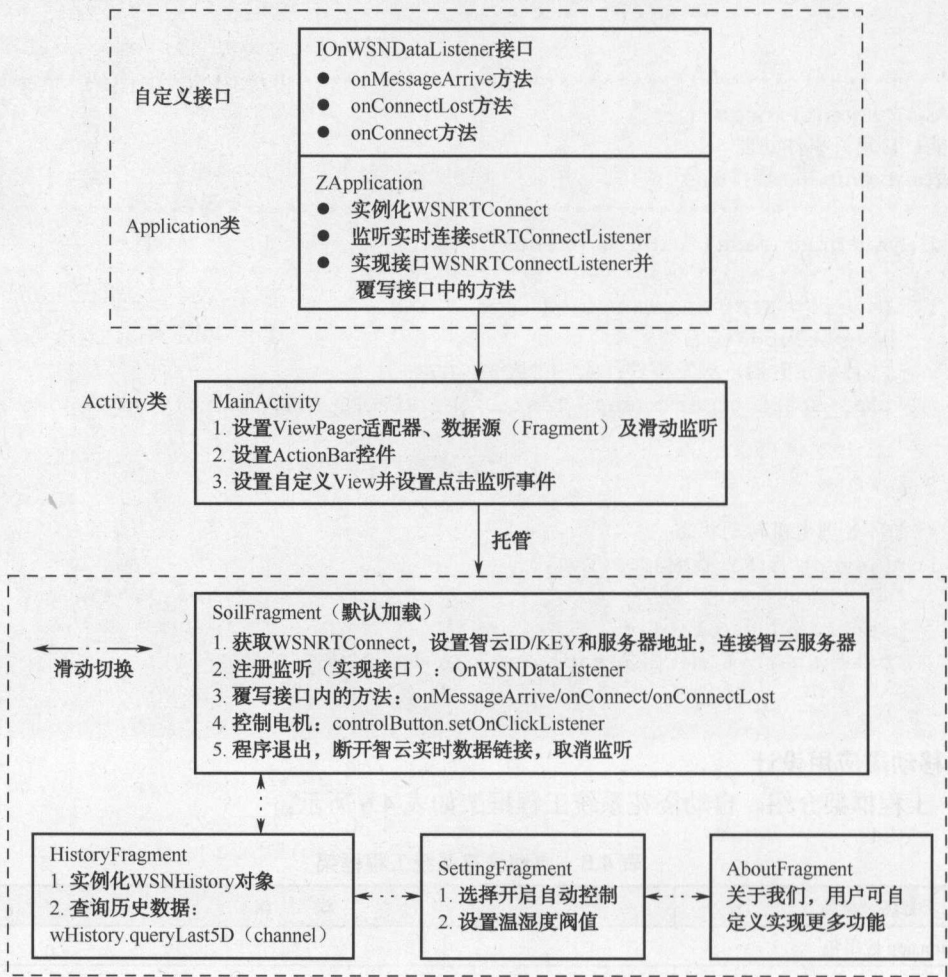


图 4.18 浇花系统程序设计逻辑



(3) 程序代码剖析。

① ZApplication 框架说明参考智能灯光控制这一章节。

② MainActivity 实现了 ViewPager 数据源设置, 各个 Fragment 模块加载, 以及自定义 View 的单击事件和 ActionBar 控件的应用, 核心源码如下所示。

```
public class MainActivity extends FragmentActivity implements
    OnPageChangeListener, OnClickListener {
    .....
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        .....
        setOverflowShowingAlways(); //显示 ActionBar 导航栏的 Overflow 按钮
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true); //使用程序图标作为 home icon
        actionBar.setDisplayHomeAsUpEnabled(true); //显示返回的箭头, 并可通过
            //onOptionsItemSelected() 进行监听, 其资源 ID 为 Android.R.id.home
        //获取 ViewPager 控件
        mViewPager = (ViewPager) findViewById(R.id.id_viewpager);
        //实例化 Fragment 及 Fragment 指示器
        initDatas();
        mViewPager.setAdapter(mAdapter); //为 mViewPager 设置适配器
        //为 mViewPager 设置页面滑动监听器, 让 Activity 去实现监听
        mViewPager.setOnPageChangeListener(this);
    }
    .....
}
```

③ 土壤温湿度实时数据采集和电机控制 (SoilFragment)。

通过 mApplication.getWSNRConnect() 获取 WSNRConnect 实例, 设置 ID/KEY 和服务地址, 通过 “mApplication.registerOnWSNDataListener (this)” 注册传感器数据监听 (实现接口 IOnWSNDataListener), 建立实时连接。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mApplication = (ZApplication) getActivity().getApplication();
    wRConnect = mApplication.getWSNRConnect();
    wRConnect.setIdKey(ID, KEY); //设置 ID/KEY
    wRConnect.setServerAddr("zhiyun360.com:28081"); //设置服务器地址
    mApplication.registerOnWSNDataListener(SoilFragment.this); //注册监听
    wRConnect.connect(); //建立实时数据服务连接
}
```

覆写接口的方法: 在 onConnect() 方法中发送查询温湿度值和电机状态的命令, 即在连接成功后立即查询温湿度值而不是一直等待底层主动上传数据。

```
@Override
public void onConnect() { //连接成功时发送查询命令
    //TODO Auto-generated method stub
    wRConnect.sendMessage(mMac, "{A0=?,A1=?}".getBytes());
    wRConnect.sendMessage(mMotorMac, "{D1=?}".getBytes());
}
```



```
Toast.makeText(getActivity(), "{A0=?,A1=?}", Toast.LENGTH_SHORT).show();
}
```

在 `onMessageArrive()` 方法中解析获取到的传感器数据：将温湿度值显示在视图中；根据电机状态来设置背景图片。

```
@Override
public void onMessageArrive(String mac, String tag, String val) {
    if (mTempMac.equalsIgnoreCase(mac)) { //过滤出温湿度传感器的数据
        System.out.println(mTempMac + tag + val);
        if (tag.equals("A0")) {
            temp = Float.parseFloat(val);
            tempTextView.setText("温度值: " + fnum.format(temp) + "°C");
        }
        if (tag.equals("A1")) {
            humi = Float.parseFloat(val);
            humiTextView.setText("湿度值: " + fnum.format(humi) + "%RH");
        }
    }
    if (mac.equalsIgnoreCase(mMotorMac)) { //过滤出电机传感器的数据
        if (tag.equals("D1")) {
            int v = Integer.parseInt(val);
            if (v == 1) {
                flag = true;
                bg.setBackgroundResource(R.drawable.jiaohua_on);
            }
            if (v == 0) {
                flag = false;
                bg.setBackgroundResource(R.drawable.jiaohua);
            }
        }
    }
}
```

单击按钮发送 “{OD1=1, D1=?}{CD1=1, D1=?}” 命令来控制电机的开关。

```
controlButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        if (flag == false) { //关闭状态
            command = "{OD1=1,D1=?}";
        } else { //打开状态
            command = "{CD1=1,D1=?}";
        }
        //发送控制电机状态的命令
        WRTConnect.sendMessage(mMotorMac, command.getBytes());
    }
});
```

④ 土壤温湿度历史数据查询 (HistoryChartFragment)：实例化 `WSNHistory`，调用 `WSNHistory` 类的历史数据查询方法来查询指定时间段内的数据并以曲线图的形式显示。在本任务中，增加了选择查询温度或者湿度的历史数据的功能，以及选择需要查询的时间段，可



单击不同的按钮来查询指定时间段的历史数据，部分源码如下所示，详细过程可参考 2.5 节中历史数据查询的流程解析。

```
case R.id.btn_week:
    mBtnWeek.setBackgroundResource(R.drawable.weekon);
    mBtnMonth.setBackgroundResource(R.drawable.month);
    mBtnMonths.setBackgroundResource(R.drawable.months);
    new getHistoryDataAsync(1).execute(new String[0]);
    Toast.makeText(getActivity(), "正在查询近一周的数据", Toast.LENGTH_SHORT).
show();
    break;
case R.id.btn_month:
    mBtnWeek.setBackgroundResource(R.drawable.week);
    mBtnMonth.setBackgroundResource(R.drawable.monthon);
    mBtnMonths.setBackgroundResource(R.drawable.months);
    new getHistoryDataAsync(2).execute(new String[0]);
    Toast.makeText(getActivity(), "正在查询近一个月的数据",
        Toast.LENGTH_SHORT).show();
    break;
case R.id.btn_months:
    mBtnWeek.setBackgroundResource(R.drawable.week);
    mBtnMonth.setBackgroundResource(R.drawable.month);
    mBtnMonths.setBackgroundResource(R.drawable.monthson);
    new getHistoryDataAsync(3).execute(new String[0]);
    Toast.makeText(getActivity(), "正在查询近三个月的数据",
        Toast.LENGTH_SHORT).show();
    break;

@Override
protected Boolean doInBackground(String... arg0) {
    //list = getHistoryData.getData(getList.mClientAPPIId, channel,
    //duration, interval, "1000", getList.mClientApiKey);
    try {
        if (i == 1) {
            historyResult = wHistory.queryLast5D(channel);
        } else if (i == 2) {
            historyResult = wHistory.queryLast1M(channel);
        } else if (i == 3) {
            historyResult = wHistory.queryLast3M(channel);
        }
        list = getList(historyResult);
    } catch (Exception e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
    }
    return true;
}
```

⑤ 自动控制设置 (SettingFragment): 当选择自动控制模式时，系统会默认设置温湿度阈值，开发者也可以自己拖动温湿度阈值滚动条来设置温度和湿度的阈值，当查询到温度值过高（大于温度阈值）或湿度值过低（小于湿度阈值）时，会自动开启电机，此时单击开关按



钮是无效的操作。

```
private void setButtonClickable() {
    if (SettingFragment.status) {
        controlButton.setClickable(false);
    } else {
        controlButton.setClickable(true);
    }
}

if (SettingFragment.status) { //开启了自动控制模式
    if (temp > SettingFragment.autoTempValue || humi < SettingFragment.
        autoHumiValue) {

        command = "{OD1=1,D1=?}";
    } else {
        command = "{CD1=1,D1=?}";
    }
    //发送查询电机状态的命令
    wRTConnect.sendMessage(mMotorMac, command.getBytes());
}
```

⑥ CheckBox 使用。在 XML 文件中定义 CheckBox 控件。

```
<CheckBox
    Android:id="@+id/checkBox"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="@string/checkbox" />
```

在 Java 文件中获取 CheckBox 控件并设置监听器，覆写其单击事件的方法。

```
//获取 CheckBox 并设置监听
autoCb = (CheckBox) v.findViewById(R.id.checkBox);
autoCb.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        //TODO Auto-generated method stub
        if (isChecked)
            status = true;
        else
            status = false;
    }
});
```

⑦ SeekBar 使用。在 XML 文件中定义 SeekBar 控件。

```
<SeekBar
    Android:id="@+id/tempSb"
    Android:layout_width="150dp"
    Android:layout_height="wrap_content" />
```

在 Java 文件中获取 SeekBar 控件并设置监听器。

```
tempSb = (SeekBar) v.findViewById(R.id.tempSb);
tempSb.setMax(100); //设置拖动条的最大值
tempSb.setOnSeekBarChangeListener(SettingFragment.this);
```

覆写监听器中的方法如下。

```
@Override
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
```



```
//TODO Auto-generated method stub
if (seekBar.equals(tempSb)) { //温度设置拖动条
    autoTempValue = progress;
    tempValue.setText("设置的温度阈值为: " + autoTempValue);
}
if (seekBar.equals(humiSb)) { //湿度设置拖动条
    autoHumiValue = progress;
    humiValue.setText("设置的湿度阈值为: " + autoHumiValue);
}
}
@Override
public void onStartTrackingTouch(SeekBar seekBar) {
    //TODO Auto-generated method stub
}
@Override
public void onStopTrackingTouch(SeekBar seekBar) {
    //TODO Auto-generated method stub
}
}
```

⑧ Spinner 使用。在 XML 文件中定义 Spinner 控件。

```
<Spinner
    Android:id="@+id/spinner"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content" />
```

在 Java 文件中获取 Spinner 控件并设置适配器。

```
spinner = (Spinner) v.findViewById(R.id.spinner);
String list[] = { "温度", "湿度" };
adapter = new ArrayAdapter<String>(getActivity(),
    Android.R.layout.simple_spinner_item, list);
adapter.setDropDownViewResource(Android.R.layout.simple_spinner_dropdown_i
tem);
spinner.setAdapter(adapter);
```

为 Spinner 控件设置监听器并覆写单击事件的方法。

```
spinner.setOnItemClickListener(new OnItemSelectedListener() {
    @Override
    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        //TODO Auto-generated method stub
        if (arg2 == 0)
            channel = channels[0];
        else
            channel = channels[1];
    }
    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        //TODO Auto-generated method stub
    }
});
```

3. Web 端应用设计

根据智云 Web 应用编程接口定义, 智能浇花系统的应用设计主要采用实时数据 API 接口和历史数据 API 接口, JS 部分代码如下。



```
var rtc = new WSNRTConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
rtc.connect(); //数据推送服务连接
$("#ConnectState").text("数据服务连接中...");
rtc.onConnect = function() { //连接成功回调函数
    rtc.sendMessage(mySensorMac1, "{A0=?}"); //向温湿度传感器发送数据获取温度值
    rtc.sendMessage(mySensorMac1, "{A1=?}"); //向温湿度传感器发送数据获取湿度值
    rtc.sendMessage(mySensorMac2, "{CD1=1,D1=?}"); //向继电器发送数据
    $("#ConnectState").text("数据服务连接成功!");
};
rtc.onConnectLost = function() { //数据服务掉线回调函数
    $("#ConnectState").text("数据服务掉线!");
};
rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
    if ((mac == mySensorMac1) && (dat.indexOf(",") == -1)) { //接收数据过滤
        var aisle = dat.substring(dat.indexOf("{") + 1, dat.indexOf("="));
        if (aisle == "A0") { //判断是否为温度值
            //将原始数据的数字部分分离出来
            dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf("}"));
            setDialData('#dial1', parseFloat(dat)); //给表盘赋值
            Temperature = dat;
        }
        if (aisle == "A1") { //判断是否为湿度值
            //将原始数据的数字部分分离出来
            dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf("}"));
            setDialData('#dial2', parseFloat(dat)); //给表盘赋值
            Humidity = dat;
        }
    }
};

$(function() {
    //选择 input 元素
    elem01 = document.querySelector('.js-min-max-start-temperature');
    init01 = new PowerRange(elem01, {
        min: -10,
        max: 50,
        start: 40,
        callback: function() { //实例化 powerRange 类并且初始化参数
            $("#rangel").text(elem01.value); //显示温度阈值
            var IsChecked = document.getElementById("checkboxid").checked;
            if (IsChecked == true) { //判断复选框是否选中
                if (parseInt(Temperature) > parseInt(elem01.value) ||
                    parseInt(Humidity) < parseInt(elem02.value))
                { //若当前温度值高于阈值或湿度值低于阈值
                    rtc.sendMessage(mySensorMac2, "{OD1=1,D1=?}"); //向继电器发
送数据
                    document.getElementById("bg").src = ("images/jh-on.gif");
                } else {
                    rtc.sendMessage(mySensorMac2, "{CD1=1,D1=?}"); //向继电器发
送数据
                }
            }
        }
    });
});
```




```

        document.getElementById("bg").src = ("images/jh-off.jpg");
    }

    });
    $("#range1").text(elem01.value);
    elem02 = document.querySelector('.js-min-max-start-humidity'); // 选择
input 元素
    init02 = new Powerrange(elem02, {
        min: 0,
        max: 50,
        start: 20,
        callback: function() { //实例化 powerange 类并且初始化参数
            $("#range2").text(elem02.value);
            var IsChecked = document.getElementById("checkboxoid").checked;
            if (IsChecked == true) {
                if (parseInt(Temperature) > parseInt(elem01.value) ||
                    parseInt(Humidity) < parseInt(elem02.value))
                { //若当前温度值高于阈值或湿度值低于阈值
                    rtc.sendMessage(mySensorMac2, "{OD1=1,D1=?}"); //向继电器发
送数据

                    document.getElementById("bg").src = ("images/jh-on.gif");
                } else {
                    rtc.sendMessage(mySensorMac2, "{CD1=1,D1=?}"); //向继电器发
送数据

                    document.getElementById("bg").src = ("images/jh-off.jpg");
                }
            }
        }
    });
    $("#range2").text(elem02.value);
});

var flag = true;
function anniu() {
    var IsChecked = document.getElementById("radioid").checked;
    if (IsChecked == true) {
        var anNiu = document.getElementById("button");
        var bG = document.getElementById("bg");
        if (flag) {
            anNiu.src = ("images/jh-an-on.png");
            bG.src = ("images/jh-on.gif");
            rtc.sendMessage(mySensorMac2, "{OD1=1,D1=?}"); //向继电器发送数据
        } else {
            anNiu.src = ("images/jh-an-off.png");
            bG.src = ("images/jh-off.jpg");
            rtc.sendMessage(mySensorMac2, "{CD1=1,D1=?}"); //向继电器发送数据
        }
        flag = !flag
    }
}

```



```
    }  
}  
  
function AutoControl() {  
    var IsChecked = document.getElementById("checkboxid").checked;  
    if (IsChecked == true) {  
        if (parseInt(Temperature) > parseInt(elem01.value) && parseInt(Humidity) <  
            parseInt(elem02.value)) { //若当前温度值高于阈值或湿度值低于阈值  
            rtc.sendMessage(mySensorMac2, "{OD1=1,D1=?}"); //向继电器发送数据  
            document.getElementById("bg").src = ("images/jh-on.gif");  
        }  
    }  
}
```

4.3.5 开发步骤

1. 搭建智云硬件环境

(1) 准备 1 台 S210 系列 Android 开发平台, 1 个土壤温湿度传感器, 1 个直流电机传感器, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到“C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples 文件夹下”。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 AutoFlowering 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入自动浇花系统主界面, 在主界面弹出“连接网关成功”消息后即表示连接到智云服务中心。

(4) 连接网关成功后会发送查询温湿度值的命令并将温湿度值在右上角显示出来, 开发者也可以单击右下角的“打开/关闭”按钮来控制电机的开关, 如图 4.19 所示。

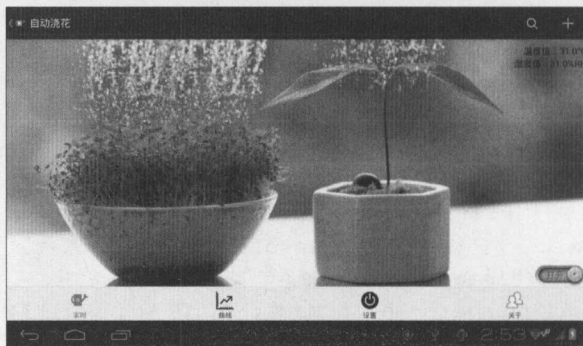


图 4.19 土壤温湿度实时数据显示



(5) 滑动或单击切换至“曲线”页面，开发者可以单击不同按钮来查询指定时间段的历史数据，也可以下拉 spinner 控件来选择查询温度或者是湿度的历史数据，如图 4.20 所示。

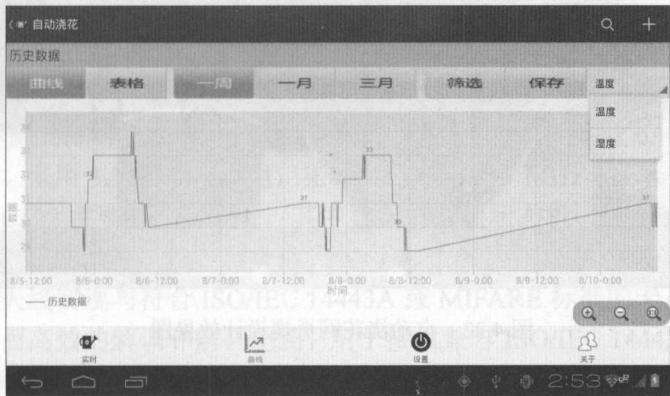


图 4.20 历史数据查询

(6) 滑动或单击切换至“设置”页面，选择自动控制模式，设置温湿度阈值，即可实现当温度过高或者湿度过低时自动开启电机的功能，设置界面如图 4.21 所示。

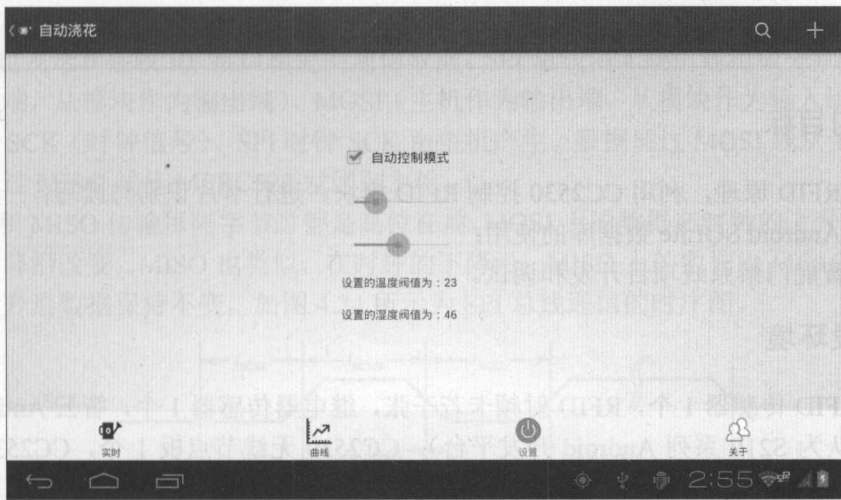


图 4.21 直流电机自动控制

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 将计算机接入互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“AutoFlowering-Web\AutoFlowering.html”，进入自动浇花系统界面，在主界面右上角显示“数据服务连接成功！”消息后即表示连接到智云服务中心，在左侧栏会实时地显示当前温湿度值，在页面下方可选择自动控制模式并设置温湿度阈值，也可以选择手动控制模式来控制电机的开关，如图 4.22 所示。



图 4.22 自动浇花网页端设计效果图

4.3.6 总结与拓展

本章任务中用到了两个传感器，使用了实时数据 API 接口和历史数据 API 接口，另外还学习了一些关于 Android UI 开发的知识。

4.4 任务 22：智能门禁系统开发（案例 10）

4.4.1 学习目标

- 了解 RFID 原理，利用 CC2530 控制 RFID 模块，进行卡片识别与读写；
- 掌握 Android SQLite 数据库的使用；
- 学会智能门禁系统项目开发和调试。

4.4.2 开发环境

硬件：RFID 传感器 1 个，RFID 射频卡若干张，继电器传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

4.4.3 原理学习

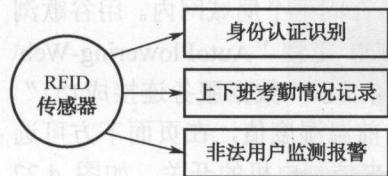


图 4.23 智能门禁系统功能模块

1. 系统设计目标

系统设计功能及目标如图 4.23 所示。

(1) 根据读取的射频卡信息确认持卡人身份，若为已注册用户，则控制继电器开门，否则不开。

(2) 记录刷卡信息和刷卡时间，存放在数据库中，作为员工考勤记录；可按需要查询数据库内容，查看员工出勤情况。

2. 业务流程分析

智能门禁系统传输过程与远程温湿度计相似，具体请参考 3.1 节中的“远程温湿度计系统”里面的内容。

3. 硬件原理

(1) 本任务的 RFID 模块 (MFRC522)，目前国内 13.56 MHz 的 RFID 读卡器芯片市场上，荷兰恩智浦公司的 Mifare 非接触读卡芯片系列中 MFRC522 系列具有低电压、低功耗、小尺寸、低成本等优点。采用 3.3 V 统一供电，工作频率为 13.56 MHz，兼容 ISO/IEC 14443A 及 Mifare 模式。MFRC522 主要包括两部分，其中数字部分由状态机、编码解码逻辑等组成；模拟部分由调制器、天线驱动器、接收器和放大器组成。MFRC522 的内部发送器无须外部有源电路即可驱动读写天线实现与符合 ISO/IEC 14443A 或 MIFARE 标准的卡片的通信。接收器模块提供了一个强健而高效的解调和解码电路，用于接收兼容 ISO/IEC 14443A 和 Mifare 的卡片信号。数字模块控制全部 ISO/IEC 14443A 帧和错误检测（奇偶和 CRC）功能。模拟接口负责处理模拟信号的调制和解调。非接触式异步收发模块配合主机处理通信协议所需要的协议。FIFO（先进先出）缓存使得主机与非接触式串行收发模块之间的数据传输变得更加快速方便。

MFRC522 支持可直接相连的各种微控制器接口类型，如 SPI、I2C 和串行 UART。在本任务中采用 SPI 总线来进行 MCU 与 RFID 模块之间的通信。SPI 接口可处理高达 10 Mbps 的数据速率。在与主机微控制器通信时，MFRC522 作为从机，接收寄存器设置的外部微控制器的数据，同时也发送和接收 RF 接口相关的通信数据。SPI 总线由 4 跟引线组成分别是 MISO（主机作为输入端，从模块作为输出端）、MOSI（主机作为输出端，从模块作为输入端）、NSS（片选信号）和 SCK（时钟信号）。SPI 时钟 SCK 由主机产生。数据通过 MOSI 线从主机传输到从机；数据通过 MISO 线从 MFRC522 发回到主机。

MOSI 和 MISO 传输每隔字节时都是高位在前。MOSI 上的数据在时钟的上升沿保持不变，在时钟的下降沿改变。MISO 也类似，在时钟的下降沿，MISO 上的数据由 MFRC522 来提供，在时钟的上升沿数据保持不变，如图 4.24 所示为 SPI 总线通信的时序图。

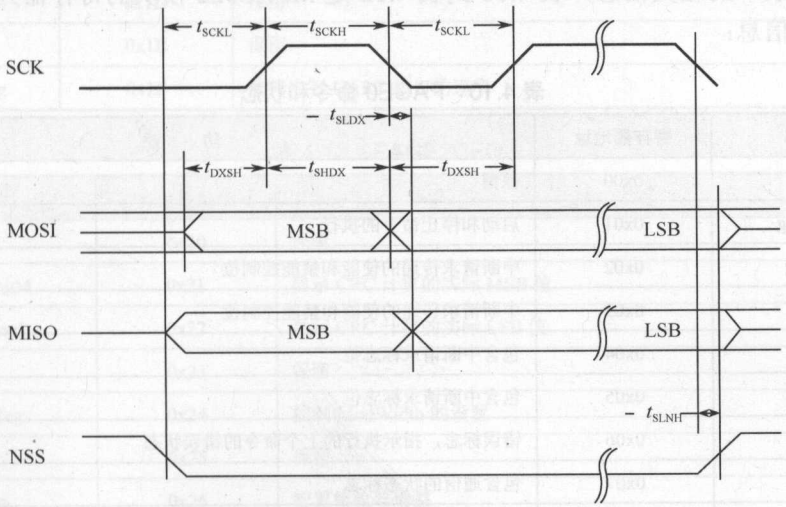


图 4.24 SPI 时序图

① 读数据。使用下面的数据结构可以通过 SPI 总线读取数据，这样可以读取 n 个字节的



数据，发送的第一个字节定义了模式本身和地址。

表 4.7 MOSI 和 MISO 的字节顺序

	字节 0	字节 1	字节 2	字节 3	...	字节 n	字节 $n+1$
MOSI	地址 0	地址 1	地址 2	地址 3	...	地址 n	00
MISO	X	数据 0	数据 1	数据 2	...	数据 $n-1$	数据 n

注意：先发送数据的最高位（MSB）

② 写数据。使用下面的数据结构可以通过 SPI 总线写数据，这样对应一个地址可以写入多达 n 个字节的数据，发送的第一个字节定义了模式本身和地址。

表 4.8 MOSI 和 MISO 的字节顺序

	字节 0	字节 1	字节 2	字节 3	...	字节 n	字节 $n+1$
MOSI	地址	数据 0	数据 1	数据 2	...	数据 $n-1$	数据 n
MISO	X	X	X	X	...	X	X

注意：先发送数据的最高位（MSB）

③ 地址字节。地址字节按下面的格式传输。第一个字节的 MSB 位设置使用的模式。MSB 位为 1 时，从 MFRC522 读书数据；MSB 位为 0 时将数据写入 MFRC522。第一个字节的 6~1 位定义地址，最后一位设置为 0。

表 4.9 地址字节格式

地址（MOSI）	位 7, MSB	位 6~位 1	位 0
字节 0	1（读）、0（写）	地址	0

通过向 MFRC522 模块的各种寄存器写入相应的值能够实现 MFRC522 模块的正常工作，包括寻卡，读取卡的相关信息，表 4.10 到表 4.12 是 MFRC522 模块的寄存器列表，以及相关的寄存器功能信息：

表 4.10 PAGE0 命令和状态

寄存器名称	寄存器地址	功 能
RFU	0x00	保留
CommandReg	0x01	启动和停止命令的执行
ComIEnReg	0x02	中断请求传递的使能和禁能控制位
DivIEnReg	0x03	中断请求传递的使能和禁能控制位
ComIrqReg	0x04	包含中断请求标志位
DivIrqReg	0x05	包含中断请求标志位
ErrorReg	0x06	错误标志，指示执行的上个命令的错误状态
Status1Reg	0x07	包含通信的状态标志
Status2Reg	0x08	包含接收器和发送器的状态标志
FIFODataReg	0x09	64 字节 FIFO 缓冲区的输入和输出
FIFOLevelReg	0x0A	指示 FIFO 中存储的字节数



续表

寄存器名称	寄存器地址	功 能
WaterLevelReg	0x0B	定义 FIFO 下溢和上溢报警的 FIFO 深度
ControlReg	0x0C	不同的控制寄存器
BitFramingReg	0x0D	面向位的帧的调节
CollReg	0x0E	RF 接口上检测到的第一个位冲突的位的位置
RFU	0x0F	保留

表 4.11 PAGE1 命令

寄存器名称	寄存器地址	功 能
RFU	0x10	保留
ModeReg	0x11	定义发送和接收的常用模式
TxModeReg	0x12	定义发送过程的数据传输速率
RxModeReg	0x13	定义接受过程中的数据传输速率
TxControlReg	0x14	控制天线驱动器管脚 TX1 和 TX2 的逻辑特性
TxAutoReg	0x15	控制天线驱动器的设置
TxSelReg	0x16	选择天线驱动器的内部源
RxSelReg	0x17	选择内部的接收器设置
RxThresholdReg	0x18	选择位译码器的阈值
DemodReg	0x19	定义解调器的设置
RFU	0x1A	保留
RFU	0x1B	保留
MifareReg	0x1C	控制 ISO 14443/MIFAR 模式中 106Kbit/s 的通信
RFU	0x1D	保留
RFU	0x1E	保留
SerialSpeedReg	0x1F	选择串行 UART 接口的速率

表 4.12 PAGE2CFG

寄存器名称	寄存器地址	功 能
RFU	0x20	保留
CRCResultRegM	0x21	显示 CRC 计算的实际 MSB 值
CRCResultRegL	0x22	显示 CRC 计算的实际 LSB 值
RFU	0x23	保留
ModWidthReg	0x24	控制 ModWidth 的设置
RFU	0x25	保留
RFCfgReg	0x26	配置接收器增益
GsNReg	0x27	选择天线驱动器管脚 TX1 和 TX2 的调制电导
CWGsCfgReg	0x28	选择天线驱动器管脚 TX1 和 TX3 的调制电导



续表

寄存器名称	寄存器地址	功 能
ModGsCfgReg	0x29	选择天线驱动器管脚 TX1 和 TX4 的调制电导
TModeReg	0x2A	定义内部定时器的设置
TPrescalerReg	0x2B	定义内部定时器的设置
TReloadRegH	0x2C	描述 16 位长的定时器重载值高位
TReloadRegL	0x2D	描述 17 位长的定时器重载值低位
TCounterValueRegH	0x2E	显示 16 位长的实际定时器值高位
TCounterValueRegL	0x2F	显示 17 位长的实际定时器值低位

在表 4.10 到表 4.12 中是各个寄存器的地址及功能列表，由于篇幅有限，通信协议请查看 MFRC522 芯片资料，MFRC522 与 CC2530 部分接口电路如图 4.25 所示。

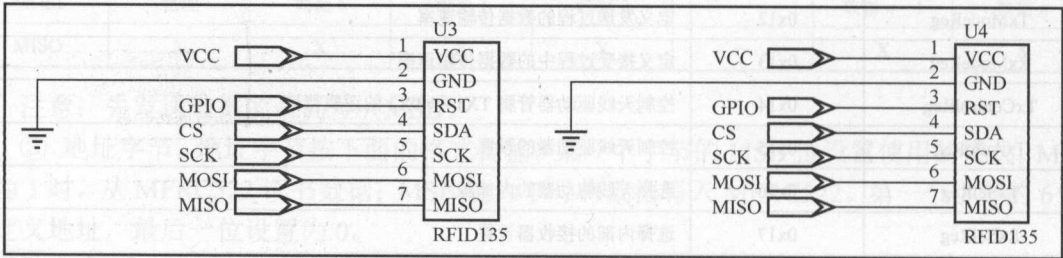


图 4.25 RFID 传感器与 CC2530 部分接口电路图

(2) 继电器工作原理说明请参考 3.2 节智能灯光控制系统相关内容。

4.4.4 开发内容

1. 硬件层驱动设计

有关继电器的详细说明请参考智能灯光控制系统这一章节相关内容，在此主要介绍 RFID 传感器的有关内容。

(1) ZXBee 智云数据通信协议。RFID 传感器的通信协议定义如表 4.13 所示。

表 4.13 相关传感器智云通信协议定义

传感器	属 性	参 数	权 限	说 明
高频 RFID	ID 卡号	A0	R	ID 卡号，字符串
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示允许识别

(2) 传感器驱动程序开发。RFID 传感器程序逻辑如图 4.26 所示。

根据 2.1 节所述，RFID 传感器属于采集类传感器，设定每隔 30 s 主动上传传感器数值。相关 ZXBee HAL 函数和部分功能函数分别如表 4.14 所示，RFID 传感器部分功能函数如表 4.15 所示。

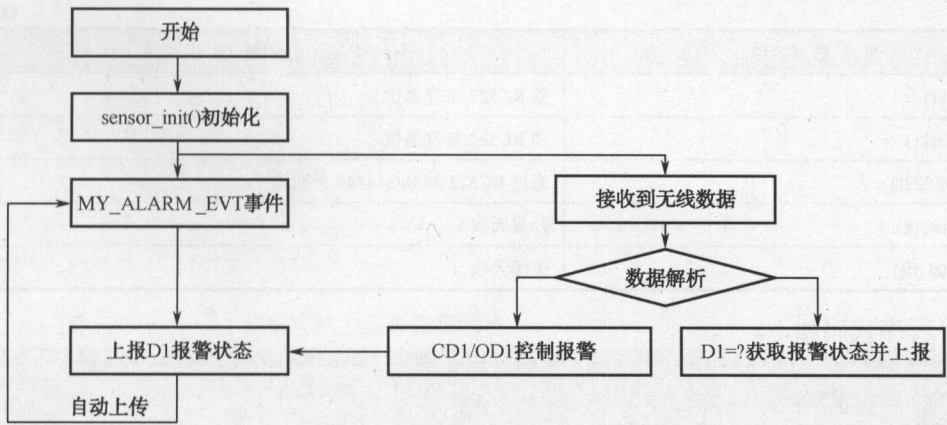


图 4.26 声光报警传感器驱动程序逻辑

表 4.14 RFID 传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	初始化传感器最基本的是配置选择寄存器和方向寄存器
updataV0()	更新主动上报的时间间隔
sensor_update()	处理主动上报的数据
sensor_check()	检测是否有可识别的射频卡贴近读写芯片
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	自定义事件处理函数，启动定时器触发事件 MY_ALARM_EVT 和 MY_CHECK_EVT

图 4.15 RFID 传感器部分功能函数

函数名称	函数说明
delay1()	延时函数
initSPIIO()	传感器初始化函数
rfid_id()	寻卡处理函数
PcdRequest()	完成寻卡操作
PcdAnticoll()	防冲撞
PcdSelect ()	选定卡片
PcdAuthState()	验证卡片密码
PcdRead()	读取 M1 卡一块数据
PcdWrite()	写数据到 M1 卡
PcdHalt()	命令卡片进入休眠状态
CalulateCRC()	用 MF522 计算 CRC16 函数
PcdReset()	复位 RC522
ReadRawRC()	读寄存器
WriteRawRC()	写寄存器



续表

函数名称	函数说明
SetBitMask()	置 RC522 寄存器位
ClearBitMask()	清 RC522 寄存器位
PcdComMF522()	通过 RC522 和 ISO14443 卡通讯
PcdAntennaOn()	开启天线
PcdAntennaOff()	关闭天线

部分程序代码如下。

```
/* ***** 宏定义 ***** */
#define SENSOR_RED      P0_1
#define SENSOR_GREEN    P1_3
#define SENSOR_BLUE     P0_6
#define SENSOR_BUZZER   P0_5
#define SENSOR_SEL      P0SEL
#define SENSOR_DIR       PODIR
#define CLKDIV ( CLKCONCMD & 0x07 )
/* ***** 全局变量 ***** */
static uint8 D0 = 1;           //默认打开主动上报功能
static uint8 D1 = 0;           //Bit0 表示声光报警状态,1 报警, 0 不报警
static uint16 V0 = 120;        //V0 设置为上报时间间隔, 默认为 120 s
static uint16 myReportInterval = 120; //上报时间间隔, 单位为 s
static uint16 myAlarmDelay = 250; //报警灯呼吸灯延时, ms
static uint8 Flag = 0;

/* *****
*名称: sensor_init()
*功能: 传感器硬件初始化
***** */
void sensor_init(void)
{
    //初始化传感器代码
    P1SEL &= ~ BV(3);
    P1DIR |= BV(3);

    SENSOR_SEL &= ~(BV(1) | BV(5) | BV(6));
    PODIR |= BV(1) | BV(5) | BV(6);

    SENSOR_RED = 1;           //RGB_LDE 中的红色,1 灭,0 亮
    SENSOR_GREEN = 1;         //RGB_LDE 中的绿色,1 灭,0 亮
    SENSOR_BLUE = 1;          //RGB_LDE 中的蓝色,1 灭,0 亮
    SENSOR_BUZZER = 0;        //蜂鸣器, 1 开, 0 关

    //启动定时器, 触发事件: MY_REPORT_EVT、MY_OFF_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10)
                                                                *1000));
    osal_start_timerEx(sapi_TaskID,MY_ALARM_EVT,myAlarmDelay); //开启报警灯循
    环呼吸灯
}
```



```

}

/*****
*名称: updateV0()
*功能: 更新 v0 的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的 v0 值
*****/
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}

/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){ //若控制编码上报允许, 则 pData 的数据包中添加控制编码数据
        len = sprintf((char*)p, "D1=%u", D1);
        p += len;
        *p++ = ',';
    }

    //将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
    if (p - pData > 1) {
        pData[0] = '{';
        p[0] = 0;
        p[-1] = '}';

        zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                                                                    AF_DEFAULT_RADIUS );
        HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
    }
}

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*****/

```



```
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest{} 发送给协
               调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);
        }
    }
    if (0 == strcmp("CD1", ptag)) {
        D1 &= ~val;
    }
    if (0 == strcmp("OD1", ptag)) {
        D1 |= val;
    }
    if (0 == strcmp("D1", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D1=%u", D1);
        }
    }
    if (0 == strcmp("V0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "V0=%u", V0);
        } else {
            updateV0(pval);
        }
    }

    return ret;
}

/*****
*名称: MyEventProcess()
*****/
```




```
*功能：自定义事件处理
*参数：event -- 事件编号
*****
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器，触发事件：MY_REPORT_EVT
        osal_start_timerEx( sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
                                                                    *1000));
    }

    //报警灯正常态呼吸灯，绿灯每隔 1s 闪烁，蜂鸣器不响
    if ( event & MY_ALARM_EVT )
    {
        if(!D1) {
            SENSOR_RED = 1;
            SENSOR_GREEN = Flag;
            SENSOR_BLUE = 1;
            SENSOR_BUZZER = 0;
        }else {
            SENSOR_RED = Flag;
            SENSOR_GREEN = 1;
            SENSOR_BLUE = 1;
            SENSOR_BUZZER = Flag;
        }
        Flag = !Flag;
        osal_start_timerEx( sapi_TaskID, MY_ALARM_EVT, 3 *!D1 *myAlarmDelay +
                                                                    myAlarmDelay);
    }
}
```

2. 移动端应用设计

(1) 工程框架介绍。智能门禁系统工程框架如表 4.16 所示。

表 4.16 智能门禁系统工程框架介绍

包名（类名）	说 明
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象，定义应用程序全局单例对象
com.zonesion.view 工具包	
ChangeColorIconWithTextView	自定义 View 的实现
com.zonesion.fragment 子模块包	
DoorCheckFragment.java	RFID 识别模块
TabFragment.java	供开发者自定义模块
com.zonesion.database 数据库包	
DBHelper.java	数据库帮助类模块



续表

包名（类名）	说 明
DBManager.java	数据库增删改查模块
com.zonesion.activity activity 包	
MainActivity.java	主 Activity

(2) 程序业务流程分析。根据 Android 应用编程接口定义，智能门禁系统的应用设计程序流程如图 4.27 所示。

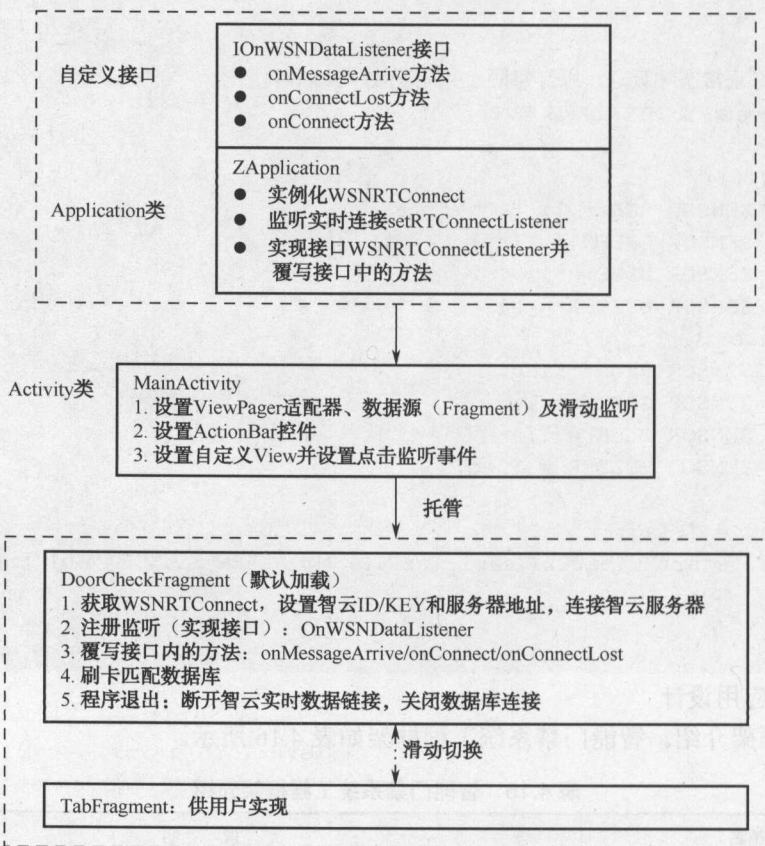


图 4.27 Android 应用程序逻辑

(3) 程序代码分析。

- ZApplication 类和 Activity 类的相关代码参考智慧窗帘控制系统这一章节相关内容；
- RFID 识别 (DoorCheckFragment)。

通过 `mApplication.getWSNRConnect()` 获取 `WSNRConnect` 实例，设置 ID/KEY 和服务地址，通过 “`mApplication.registerOnWSNDataListener (this)`” 注册传感器数据监听（实现接口 `IOOnWSNDataListener`），建立实时连接。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```



```

mApplication = (ZApplication) getActivity().getApplication();
wRTConnect = mApplication.getWSNRConnect();
wRTConnect.setIDKey(ID, KEY); //设置 ID/KEY
wRTConnect.setServerAddr("zhiyun360.com:28081"); //设置服务器地址
mApplication.registerOnWSNDataListener(SoilFragment.this); //注册监听
wRTConnect.connect(); //建立实时数据服务连接
}

```

覆写接口的方法：在 `onMessageArrive()` 方法中解析获取到的传感器数据，将卡号显示在视图中；若数据库中有对应的卡号和用户名，则为合法用户，打开继电器；否则为非法用户，不打开继电器。

```

@Override
public void onMessageArrive(String mac, String tag, String val) {
    if (rfid_mac.equalsIgnoreCase(mac)) {
        if (tag.equals("A0")) {
            tvCard.setText(val);
            tvWelcome.setEnabled(true);
            //在数据库中查询卡号,返回用户名
            String db_user_name = dbManager.query(val);
            //如果用户名不为空,则发送打开命令,并在文本显示控件上提示该用户打卡成功
            if (!db_user_name.equals("")) {
                mWSNRConnect.sendMessage(relay_mac, "{OD1=1,D1=?}".getBytes());
                tvWelcome.setText(db_user_name + "打卡成功");
            } else { //如果用户名为空,则提示该卡号未注册
                tvWelcome.setText(val + "未注册");
            }
            //如果收到主动上报的 A0=0 消息,文本显示控件文本清空,事件功能失效,并发送关闭命令
            if (val.equals("0")) {
                tvCard.setText("");
                tvWelcome.setText("");
                tvWelcome.setEnabled(false);
                mWSNRConnect.sendMessage(relay_mac,
                    "{CD1=1,D1=?}".getBytes());
            }
        }
    }
}

```

Android 开发中使用 SQLite 数据库：Activites 可以通过 Content Provider 或者 Service 访问一个数据库。

① 创建数据库。Android 不自动提供数据库。在 Android 应用程序中使用 SQLite 数据库，必须自己创建数据库，然后创建表索引，填充数据。Android 提供了 `SQLiteOpenHelper` 帮助创建一个数据库，只要继承 `SQLiteOpenHelper` 类，就可以轻松的创建数据库。`SQLiteOpenHelper` 类根据开发应用程序的需要，封装了创建和更新数据库使用的逻辑。`SQLiteOpenHelper` 的子类，需要实现以下三个方法。

构造函数，调用父类 `SQLiteOpenHelper` 的构造函数。这个方法需要四个参数：上下文环境（例如一个 Activity），数据库名字，一个可选的游标工厂（通常是 Null），一个代表你正在使用的数据库模型版本的整数。

`onCreate()` 方法，它需要一个 `SQLiteDatabase` 对象作为参数，根据需要对这个对象填充表



和初始化数据。

`onUpgrade()` 方法，它需要三个参数，一个 `SQLiteDatabase` 对象，一个旧的版本号和一个新的版本号，把一个数据库从旧的模型转变到新的模型。

```
public class DBHelper extends SQLiteOpenHelper {
    DBHelper(Context context, String name, CursorFactory cursorFactory, int
version)
    {
        super(context, name, cursorFactory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //创建数据库后，对数据库的操作
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //更改数据库版本的操作
    }
}
```

② 创建表，插入数据。调用 `getWritableDatabase()` 方法，得到一个 `SQLiteDatabase` 实例，使用这个对象，可以查询或者修改数据库。

```
mDBHelper = new DBHelper(context);           //创建数据库助手对象
db = mDBHelper.getWritableDatabase();         //获取可写的数据库
```

创建表：调用 `SQLiteDatabase` 的 `execSQL()` 方法来执行 DDL 语句。

```
db.execSQL("create table if not exists users"+
            "(_id integer primary key autoincrement, card_num string not null,
            user_name string not null);");
```

给表添加数据：使用 `SQLiteDatabase` 对象的 `insert()` 方法，把 SQL 语句的一部分作为参数。

```
ContentValues cv=new ContentValues();
cv.put(Constants.TITLE, "example title");
cv.put(Constants.VALUE, SensorManager.GRAVITY_DEATH_STAR_I);
db.insert("mytable", getNullColumnHack(), cv);
```

③ 查询数据库。使用 `rawQuery()` 方法直接调用 SQL SELECT 语句，返回值是一个 `cursor` 对象，这个对象的方法可以迭代查询结果。

```
String sql = "select *from users where card_num =" + "'" + card_num + "'";
String user_name = "";
Cursor c = db.rawQuery(sql, null);
while (c.moveToNext()) {
    user_name = c.getString(c.getColumnIndex("user_name"));
}
c.close();
return user_name;
```

3. Web 端应用设计

根据 Web 应用编程接口定义，智能门禁系统的应用设计主要采用实时数据 API 接口，JS 部分代码如下。



```

<script type="text/javascript">
var myZCloudID = "12345678";           //序列号
var myZCloudKey = "12345678";         //密钥
var mySensorMac1 = "00:12:4B:00:02:60:E5:26"; //传感器的 MAC 地址 (RFID 传感器)
var mySensorMac2 = "00:12:4B:00:03:A7:E1:17"; //传感器的 MAC 地址 (继电器)
var rtc = new WSNRTCConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象

rtc.connect();                         //数据推送服务连接
$("#ConnectState").text("数据服务连接中...");

rtc.onConnect = function () { //连接成功回调函数
    rtc.sendMessage(mySensorMac1, "{OD0=1,D0=?}"); //向 RFID 传感器发送数据 (开启)
    rtc.sendMessage(mySensorMac1, "{A0=?}"); //向 RFID 传感器发送数据 (获取卡号)
    rtc.sendMessage(mySensorMac2, "{CD1=1,D1=?}"); //向继电器发送数据 (关)
    $("#ConnectState").text("数据服务连接成功!");
};

rtc.onConnectLost = function () { //数据服务掉线回调函数
    $("#ConnectState").text("数据服务掉线!");
};

rtc.onmessageArrive = function (mac, dat) { //消息处理回调函数
    if ((mac == mySensorMac1) && (dat.indexOf(",") == -1)) { //接收数据过滤
        //将原始数据的数字部分分离出来
        dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf(")"));
        if (dat.length == 8 ) { //检测是否有用户
            //判断是否为合法用户
            if (dat == "93BAE901" || dat == "7338E601" || dat == "83D7E901") {
                $("#tip").text("欢迎! ID: "+dat);
                document.getElementById("door").src = ("images/on.png");
                rtc.sendMessage(mySensorMac2, "{OD1=1,D1=?}"); //向继电器发送数据
            (开)
            }
            else {
                $("#tip").text("非法用户!");
            }
        }
        else{
            $("#tip").text("等待用户");
            document.getElementById("door").src = ("images/off.png");
            rtc.sendMessage(mySensorMac2, "{CD1=1,D1=?}"); //向继电器发送数据 (关)
        }
    }
};
</script>

```

4.4.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个 RFID 传感器无线节点、1 个继电器传感器, 按照任务 3 的方法设置节点板跳线为模式一。



(2) 打开传感器驱动工程：将本任务 SensorHalExamples 下所有文件夹复制到 “C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples” 文件夹下。

(3) 分别打开协调器和传感器工程，编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中，同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件，组成智云无线传感网络，并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 DoorCheck 工程，并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入系统默认加载的 Fragment (DoorCheckFragment)，进入 RFID 射频卡识别界面，在界面弹出 “连接网关成功” 消息后即表示连接到智云服务中心。将射频卡靠近 RFID 传感器，当读取到射频卡的卡号时，会显示在屏幕左侧上，由于最开始没有想数据库中增加卡号及其对应的用户名，所以在屏幕上方的 TextView 中会显示 “未注册” 的信息，如图 4.28 所示。

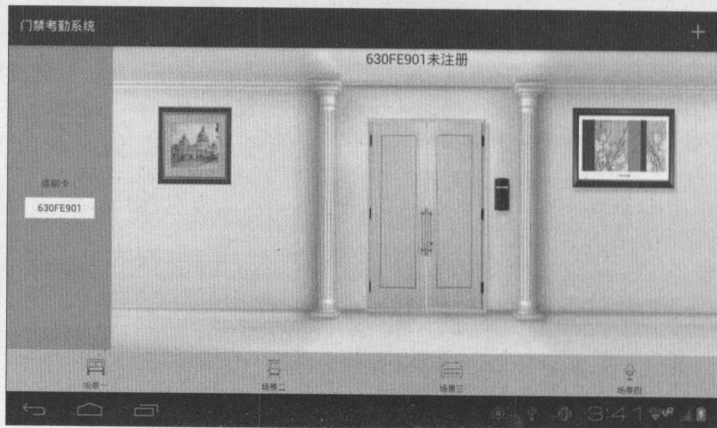


图 4.28 智能考勤 RFID 识别界面

(4) 长按屏幕上方的 TextView 控件，会弹出对话框，提示输入与此射频卡对应的用户名，如我是员工 1，如图 4.29 所示。

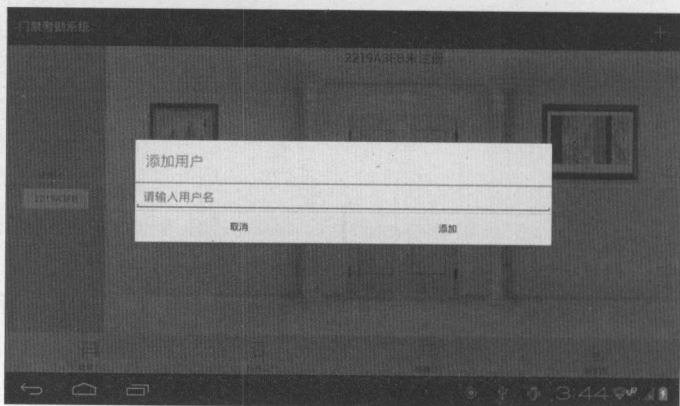


图 4.29 添加对应的用户名



(5) 当开发者添加至数据库成功后,再次刷卡,TextView 会显示“我是员工 1 打卡成功”,如图 4.30 所示。

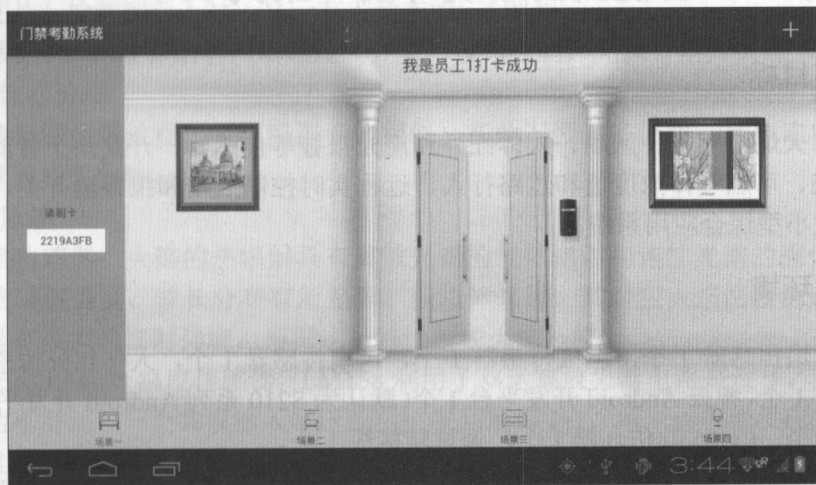


图 4.30 打卡成功

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址(若在局域网内使用,则设置为智云 Android 开发平台的 IP)和智云 ID/KEY。

(2) 电脑接入到互联网,或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器(或支持 HTML5 技术的 IE10 以上版本浏览器)运行 Web 工程“SmartRFID-Web\SmartRFID.html”,进入智能门禁系统界面,在主界面右上角显示“数据服务连接成功!”消息后即表示连接到智云服务中心,在页面正上方会显示读取的 RFID 射频卡卡号,若为合法用户(数据库中有此卡号及其对应的用户名),则开门(打开继电器),如图 4.31 所示。



图 4.31 智能门禁系统 Web 端

4.4.6 总结与拓展

本任务使用的是 RFID 传感器,用于读取射频卡的信息(例如卡号等);另外还涉及了 Android 中数据库的基本使用。



4.5 任务 23：智能安防系统开发（案例 11）

4.5.1 学习目标

学会利用火焰、燃气、风扇、人体红外和声光报警等传感器，开发一个智能安防系统，能够监测火焰、可燃气体浓度值和过路行人，远程实时控制风扇和报警器开关，学会使用多个传感器开发小型综合应用系统。

4.5.2 开发环境

硬件：火焰传感器 1 个，燃气传感器 1 个，风扇传感器 1 个，人体红外传感器 1 个，声光报警传感器 1 个，智云 Android 开发平台 1 个(默认为 S210 系列 Android 开发平台)，CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools (Android 集成开发环境)。

4.5.3 原理学习

1. 系统设计目标

系统设计功能及目标如图 4.32 所示。

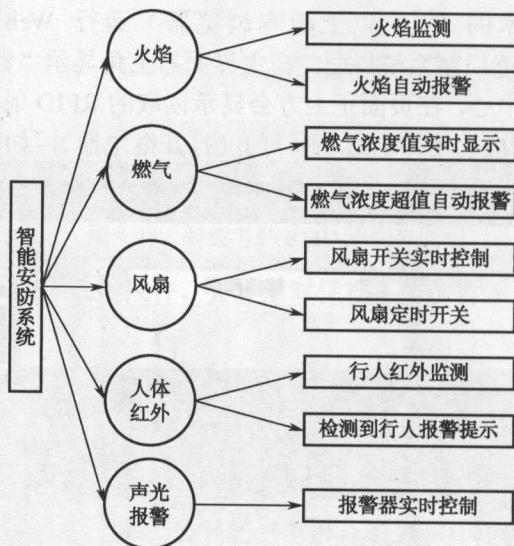


图 4.32 智能安防系统功能模块

(1) 通过火焰传感器检测火焰，无火焰时，在 Android 移动客户端显示无火焰的图片；有火焰时显示有火焰的图片。

(2) 通过燃起传感器检测周围环境的燃气浓度值并实时显示在 Android 移动客户端中。

(3) 能够在 Android 移动客户端通过“开关”按钮控制风扇传感器，远程实时控制风扇开



关，默认情况下，风扇是静止的，风扇工作时显示风扇叶子快速转动的 gif 图片。

(4) 通过人体红外传感器对行人进行检测, 在没有检测到行人时, 在 Android 移动客户端显示无人的图片; 检测到行人时, 显示有人动态的 gif 图片。

(5) 能够在 Android 移动客户端通过“开关”按钮控制声光报警器开关。

2. 业务流程分析

智能安防系统传输过程与远程温湿度计相似，在此不做过多说明，具体请参考第远程温湿度计这一章节相关内容。

3. 硬件原理

(1) 火焰传感器: 火焰的热辐射具有离散光谱的气体辐射和连续光谱的固体辐射, 不同燃烧物的火焰辐射强度、波长分布有所差异, 但总体来说, 其对应火焰温度的近红外波长域及紫外光域具有很大的辐射强度, 根据这种特性制成火焰传感器。

本任务采用 LM158 温度传感器, LM158 利用了双运算放大器电路来设计, 其特性包括以下几点: 低功率消耗; 一个共模输入电压范围扩展到地和 V_{EE} ; 单一供应或供应分流; 采用了 MC1558 双运算放大器插脚引线, LM158 系列功耗相当于 LM124 的一半。

这些放大器有几个明显的优势，他们在供应电压可以低至 3.0 V 或高达 32 V，其中静态电流约为 MC1741 的 1/5，共模输入范围包括负供给，从而消除外部偏置组件的必要性在许多应用程序中，输出电压范围还包括负电源电压。

本任务中的火焰传感器当检测到火焰时输出一个高电平, 未检测到火焰时输出一个低电平, 火焰传感器与 CC2530 部分接口电路如图 4.33 所示。

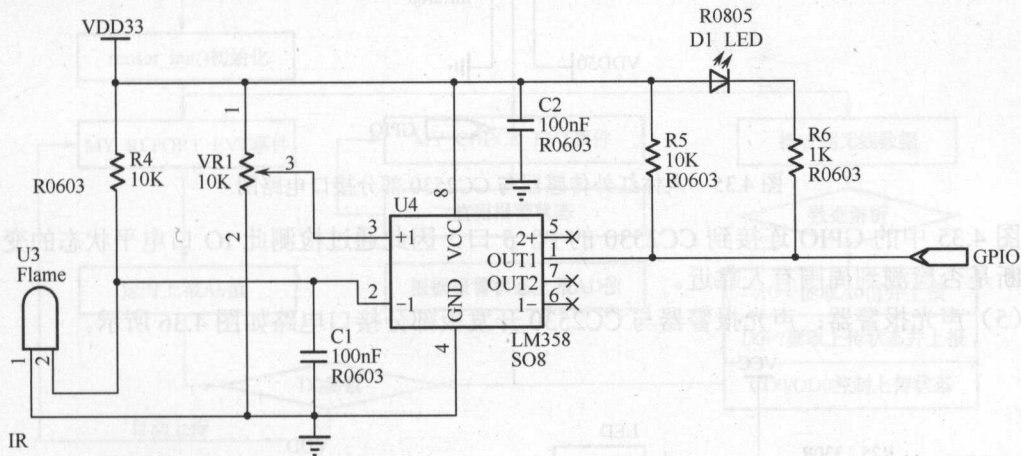


图 4.33 火焰传感器与 CC2530 部分接口电路图

图 4.33 中的 GPIO 引脚连接到 CC2530 的 P0 5 口, 可以直接读取此 IO 口输入的信号。

(2) 可燃气体传感器: 可燃气体传感器原理说明请参考厨房燃气监测系统这一章节相关内容。

(3) 风扇传感器: 风扇传感器与 CC2530 部分接口电路如图 4.32 所示。

图 4.34 中的 GPIO 引脚连接到了 CC2530 的 P0_5 口, 可以直接读取此 IO 口输入的信号; MISO 引脚接到 CC2530 的 P0_6 口。

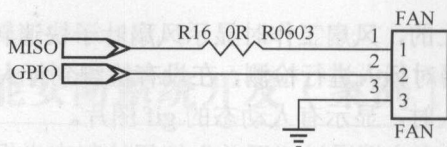


图 4.34 风扇传感器与 CC2530 部分接口电路图

(4) 人体红外传感器：普通体会发射 $10\ \mu\text{m}$ 左右的特定波长红外线，用专门设计的传感器就可以针对性的检测这种红外线的存在与否，当人体红外线照射到传感器上后，因热释电效应将向外释放电荷，后续电路经检测处理后就能产生控制信号。

本任务使用的 HC-SR501 传感器是基于红外线技术的自动控制模块，广泛应用于各类自动感应电器设备。HC-SR501 的核心控制模块是采用稳定性好、可靠性强、灵敏度高且超低功耗的 LH1788 探头的自动控制模块，LH1788 更是在红外技术下衍生而来的器件，超低的驱动电压便是该模块的优势所在。

人体红外传感器检测到有人体活动时，其输出的 IO 值发生变化。当传感器模块检测到有人入侵时，会返回一个高电平信号，无人入侵时，返回一个低电平信号，通过读取 IO 口的状态判断是否有人体活动，人体红外传感器与 CC2530 开发板部分接口电路如图 4.35 所示。

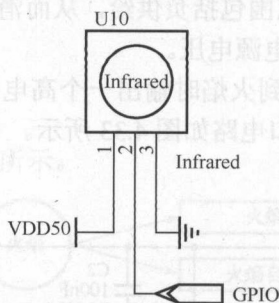


图 4.35 人体红外传感器与 CC2530 部分接口电路图

图 4.35 中的 GPIO 连接到 CC2530 的 P0_5 口，因此通过检测此 IO 口电平状态的变化，可判断是否检测到周围有人靠近。

(5) 声光报警器：声光报警器与 CC2530 开发板部分接口电路如图 4.36 所示。

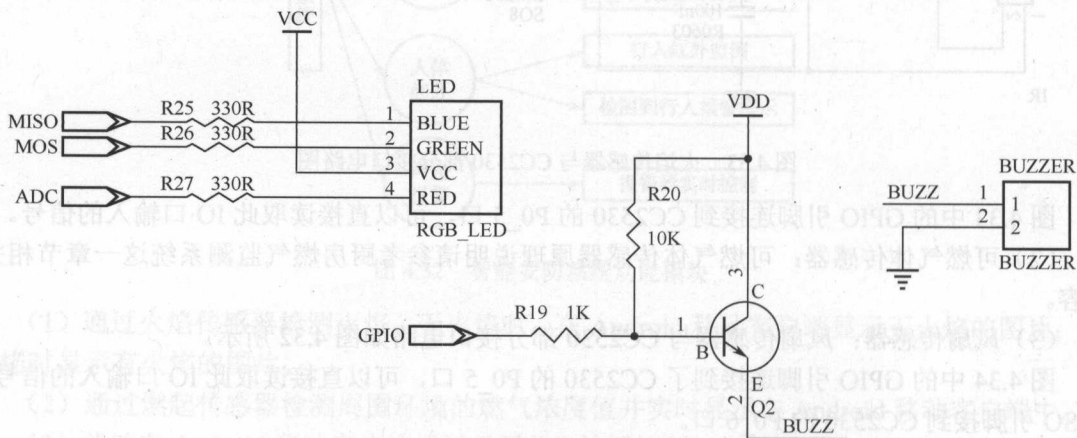


图 4.36 声光报警传感器与 CC2530 部分接口电路图

图 4.36 中的 GPIO 连接到 CC2530 的 P0_5 口, ADC 连接到 CC2530 的 P0_1 口, MISO 连接到 CC2530 的 P0_6 口, MOSI 连接到 CC2530 的 P1_3 口。

4.5.4 开发内容

1. 硬件层驱动设计

(1) ZXBee 智云数据通信协议。根据 2.1 节介绍, 定义火焰传感器、燃气传感器、人体红外传感器、风扇传感器和声光报警传感器的通信协议如表 4.17 所示。

表 4.17 相关传感器智云通信协议定义

传感器	属性	参数	权限	说 明
可燃气体	数值	A0	R	数值, 燃气浓度值
火焰/人体红外	数值	A0	R	数值, 0 或者 1 变化
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态
风扇/声光报警	电源开关	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示电源状态, OD1 为开、CD1 为关

(2) 传感器驱动程序开发。可燃气体传感器的驱动开发请参考可燃气体检测系统这一章节相关内容。火焰传感器与人体红外传感器类似, 以火焰传感器为例进行分析。火焰传感器的程序逻辑如图 4.37 所示。

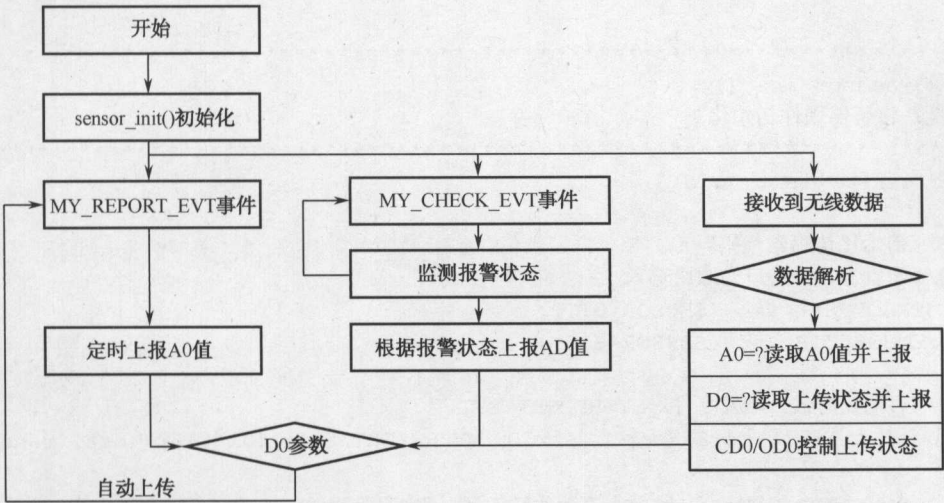


图 4.37 火焰传感器驱动程序流程分析

根据 2.1 节所述, 火焰传感器属于采集类传感器, 设定每隔 120 s 主动上报传感器数值。相关传感器 ZXBee HAL 函数如表 4.18 所示。

表 4.18 相关传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	初始化传感器
updateV0()	更新上报时间
updateA0()	更新 A0 的值



续表

函数名称	函数说明
sensor_check()	传感器控制函数
sensor_update()	上报采集到的数据
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	事件处理函数，启动定时器触发 MY_REPORT_EVT 和 MY_ALARM_EVT 事件

部分程序代码如下。

```
/ *****宏定义*****/
#define SENSOR_PORT P0
#define SENSOR_SEL POSEL
#define SENSOR_DIR PODIR
#define SENSOR_BIT 0x20
#define SENSOR_PIN P0_5

/ *****全局变量*****/
static uint8 D0 = 1; //默认打开主动上报功能
static uint8 A0 = 0; //报警状态值
static uint16 V0 = 120; //V0 设置为上报时间间隔，默认为 120 s
static uint16 myReportInterval = 120; //上报时间间隔，单位为 s
static uint16 Flag = 0; //报警标识

/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    //配置 P0_5 端口为通用输入 IO
    SENSOR_SEL &= ~(SENSOR_BIT);
    SENSOR_DIR &= ~(SENSOR_BIT);

    //启动定时器，触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10
                                                                *1000));
    osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, (uint16)((osal_rand()%10
                                                                *1000));
}

/*****
*名称: updateV0()
*功能: 更新 V0 的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的 V0 值
*****/
uint16 updateV0(char *val)
{

```




```
//将字符串变量 val 解析转换为整型变量赋值
myReportInterval = atoi(val);
V0 = myReportInterval;

return V0;
}

/*****
*名称: updateA0()
*功能: 更新 A0 的值
*参数: 无
*返回: A0 -- 返回更新后的 A0 值
*****/
uint8 updateA0(void)
{
    A0 = !SENSOR_PIN;
    return A0;
}

/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){ //若报警值上报允许, 则 pData 的数据包中添加报警值数据
        updateA0();
        len = sprintf((char*)p, "A0=%u", A0);
        p += len;
        *p++ = ',';
    }

    //将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
    if (p - pData > 1) {
        pData[0] = '{';
        p[0] = 0;
        p[-1] = '}';

        zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                           AF_DEFAULT_RADIUS );
        HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
    }
}
```



```
/******
*名称: sensor_check()
*功能: 监测报警值
******/
void sensor_check(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    int len;
    uint8 lastA0 = 0;

    if((D0 & 0x01) == 1){
        lastA0 = A0; //记录上次 A0 的值
        updateA0();

        //当监测到维持高电平状态, 上报报警值 A0=1
        if (A0 == 1) {
            if(Flag % 3 == 0){ //每 3 秒报警一次
                len = sprintf((char*)pData, "{A0=%u}", A0);
                zb_SendDataRequest(0, cmd, len, (uint8*)pData, 0, AF_ACK_REQUEST,
                                   AF_DEFAULT_RADIUS); //发送数据到协调器
                HalLedSet(HAL_LED_1, HAL_LED_MODE_BLINK); //通信 LED 闪烁一次
            }
            Flag++;
        }
        //当监测到维持低电平状态, 上报清除报警状态 A0=0
        else if ((Flag != 0) && (lastA0 == 0) && (A0 == 0)) {
            len = sprintf((char*)pData, "{A0=%u}", A0);
            zb_SendDataRequest(0, cmd, len, (uint8*)pData, 0, AF_ACK_REQUEST,
                               AF_DEFAULT_RADIUS); //发送数据到协调器
            HalLedSet(HAL_LED_1, HAL_LED_MODE_BLINK); //通信 LED 闪烁一次
            Flag = 0;
        }
    }
}

/******
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据
*返回: ret -- pout 字符串长度
******/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
```



```

val = atoi(pval);

//控制命令解析
if (0 == strcmp("CD0", ptag)) {
    D0 &= ~val;
}
if (0 == strcmp("OD0", ptag)) {
    D0 |= val;
}
if (0 == strcmp("D0", ptag)) {
    if (0 == strcmp("?", pval)) {
        ret = sprintf(pout, "D0=%u", D0);
    }
}
if (0 == strcmp("A0", ptag)) {
    if (0 == strcmp("?", pval)) {
        updateA0();
        ret = sprintf(pout, "A0=%u", A0);
    }
}
if (0 == strcmp("V0", ptag)) {
    if (0 == strcmp("?", pval)) {
        ret = sprintf(pout, "V0=%u", V0);
    } else {
        updateV0(pval);
    }
}
return ret;
}

/*****
*名称: MyEventProcess()
*功能: 解析收到的控制命令
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)(myReportInterval
                                                                    *1000));
    }
    if (event & MY_CHECK_EVT) {
        sensor_check();
        //启动定时器, 触发事件MY_CHECK_EVT, 定时查询报警值
        osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, 1000);
    }
}

```

风扇传感器的驱动开发与继电器类似, 请参考智能灯光控制系统这一章节相关内容。声



光报警传感器程序逻辑如图 4.38 所示。

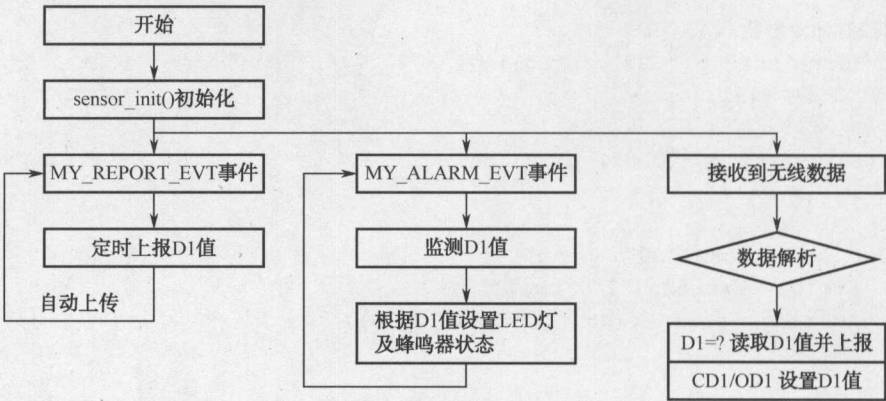


图 4.38 声光报警器驱动程序逻辑

根据 2.1 节所述，声光报警传感器属于控制类传感器，设定每隔 120 s 主动上报传感器数值。但声光报警传感器比继电器增加了一个 MY_ALARM_EVT 事件，在响应事件时，根据 D1 的值控制 LED 灯及蜂鸣器的状态。相关传感器 ZXBee HAL 函数如表 4.19 所示。

表 4.19 相关传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	初始化传感器
sensor_control()	传感器控制函数
sensor_update()	上报采集到的数据
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	事件处理函数，启动定时器触发 MY_REPORT_EVT 和 MY_ALARM_EVT 事件

部分程序代码如下。

```

/ *****宏定义*****/
#define SENSOR_RED      P0_1
#define SENSOR_GREEN    P1_3
#define SENSOR_BLUE     P0_6
#define SENSOR_BUZZER   P0_5
#define SENSOR_SEL      POSEL
#define SENSOR_DIR      PODIR
#define CLKDIV ( CLKCONCMD & 0x07 )

/ *****全局变量*****/
static uint8 D0 = 1;           //默认打开主动上报功能
static uint8 D1 = 0;           //Bit0 表示声光报警状态,1 报警, 0 不报警
static uint16 V0 = 120;        //V0 设置为上报时间间隔, 默认为 120 s
static uint16 myReportInterval = 120; //上报时间间隔, 单位为 s
static uint16 myAlarmDelay = 250; //报警灯呼吸灯延时, 毫秒
static uint8 Flag = 0;

/*****
*名称: sensor_init()

```



*功能: 传感器硬件初始化

```

*****/
void sensor_init(void)
{
    //初始化传感器代码
    P1SEL &= ~ BV(3);
    P1DIR |= BV(3);

    SENSOR_SEL &= ~(BV(1) | BV(5) | BV(6));
    PODIR |= BV(1) | BV(5) | BV(6);

    SENSOR_RED = 1;                //RGB_LDE 中的红色,1 灭,0 亮
    SENSOR_GREEN = 1;              //RGB_LDE 中的绿色,1 灭,0 亮
    SENSOR_BLUE = 1;               //RGB_LDE 中的蓝色,1 灭,0 亮
    SENSOR_BUZZER = 0;             //蜂鸣器, 1 开, 0 关

    //启动定时器, 触发事件: MY_REPORT_EVT、MY_OFF_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10)
                                                                *1000));
    osal_start_timerEx(sapi_TaskID, MY_ALARM_EVT, myAlarmDelay); //开启报警灯循
    环呼吸灯
}

```

```

/*****
*名称: updateV0()
*功能: 更新 v0 的值
*参数: *val -- 待更新的变量
*返回: v0 -- 返回更新后的 v0 值
*****/

```

```

uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}

```

```

/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/

```

```

void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

```

//根据 D0 的位状态判定需要主动上报的数值



```
if ((D0 & 0x01) == 0x01) { //若控制编码上报允许, 则 pData 的数据包中添加控制编码数据
    len = sprintf((char*)p, "D1=%u", D1);
    p += len;
    *p++ = ',';
}

//将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
if (p - pData > 1) {
    pData[0] = '{';
    p[0] = 0;
    p[-1] = '}';

    zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
        AF_DEFAULT_RADIUS );
    HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
}

}

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest{} 发送给协
        调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);
        }
    }
    if (0 == strcmp("CD1", ptag)) {
        D1 &= ~val;
    }
}
```




```

if (0 == strcmp("OD1", ptag)) {
    D1 |= val;
}
if (0 == strcmp("D1", ptag)) {
    if (0 == strcmp("?", pval)) {
        ret = sprintf(pout, "D1=%u", D1);
    }
}
if (0 == strcmp("V0", ptag)) {
    if (0 == strcmp("?", pval)) {
        ret = sprintf(pout, "V0=%u", V0);
    }else{
        updateV0(pval);
    }
}
return ret;
}

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件 MY_REPORT_EVT
        osal_start_timerEx( sapi_TaskID, MY_REPORT_EVT, (uint16) (myReportInterval
                                                                    *1000));
    }

    //报警灯正常态呼吸灯, 绿灯每隔 1s 闪烁, 蜂鸣器不响
    if ( event & MY_ALARM_EVT )
    {
        if(!D1) {
            SENSOR_RED = 1;
            SENSOR_GREEN = Flag;
            SENSOR_BLUE = 1;
            SENSOR_BUZZER = 0;
        }else {
            SENSOR_RED = Flag;
            SENSOR_GREEN = 1;
            SENSOR_BLUE = 1;
            SENSOR_BUZZER = Flag;
        }
        Flag = !Flag;
        osal_start_timerEx( sapi_TaskID, MY_ALARM_EVT, 3 *!D1 *myAlarmDelay +
                                                                    myAlarmDelay);
    }
}

```



2. 移动端应用设计

(1) 工程框架介绍。智能安防系统工程框架如表 4.20 所示。

表 4.20 智能安防系统工程框架介绍

包名（类名）	说 明
com.zonesion.activity 主 Activity 包	
MainActivity.java	主 Activity，建立智云连接，实例化 Fragment
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象，定义应用程序全局单例对象
com.zonesion.gif gif 包	
GIFMovieView.java	自定义 View，用于播放 Gif 动画
com.zonesion.tool 包	
ChangeColorIconWithTextView.java	自定义 View
com.zonesion.ui 子模块 Fragment 包	
AlarmFragment.java	火焰、燃气、声光报警的控制报警模块
FannerFragment.java	风扇控制模块
InfraredFragment.java	人体红外模块

(2) 程序业务流程分析。根据 2.1 节智云 Android 应用编程接口定义，智能安防系统的应用设计主要采用实时数据 API 接口，程序流程如图 4.39 所示。

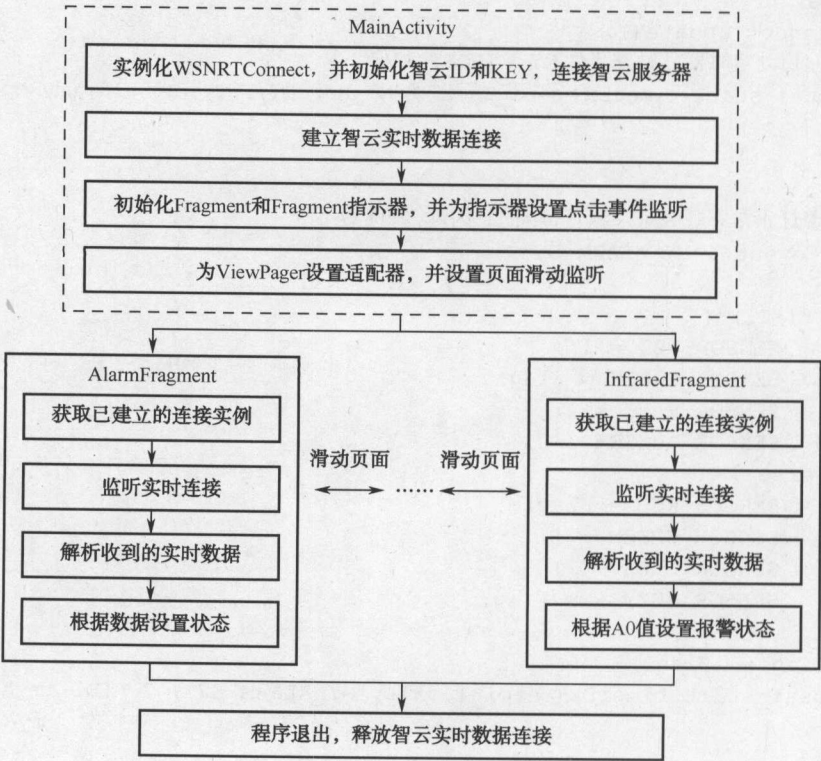


图 4.39 Android 应用程序逻辑



(3) 程序代码剖析。以人体红外传感器为例对程序代码进行剖析，其他传感器与之类似，可参考图 4.39 自行分析。

① MainActivity 初始化工作。实例化 WSNRTCConnect，并初始化智云 ID 和 KEY，连接智云服务器（局域网内使用：ID/KEY 可任意填写，注意每个 ID 不能相同，服务地址为智云 Android 开发平台的 IP 地址。若链接到互联网访问，需要填写正确的 ID/KEY/服务地址）。

```
mApplication = (ZApplication) getApplication();
wRTCConnect = mApplication.getWSNRTCConnect();
wRTCConnect.setIdKey(id, key); //初始化智云 ID 和 KEY
wRTCConnect.setServerAddr("zhiyun360.com:28081"); //设置服务器地址
```

建立智云实时数据链接。

```
wRTCConnect.connect();
```

实例化 Fragment 和 ViewPager 适配器。

```
private void initDatas() {
    //实例化 Fragment
    AlarmFragment flameFragment = new AlarmFragment();
    FannerFragment fannerFragment = new FannerFragment();
    InfraredFragment infraredFragment = new InfraredFragment();

    mFragments.add(flameFragment);
    mFragments.add(fannerFragment);
    mFragments.add(infraredFragment);

    mAdapter = new FragmentPagerAdapter(getSupportFragmentManager()) {
        @Override
        public int getCount() {
            //TODO Auto-generated method stub
            return mFragments.size();
        }

        @Override
        public Fragment getItem(int arg0) {
            //TODO Auto-generated method stub
            return mFragments.get(arg0);
        }
    };

    initFragmentIndicator(); //初始化 Fragment 指示器,即 ChangeColorIconWithTextView
}
```

初始化 Fragment 指示器。

//初始化 Fragment 指示器

```
private void initFragmentIndicator() {
    ChangeColorIconWithTextView one = (ChangeColorIconWithTextView)
        findViewById(R.id.id_indicator_one);
    ChangeColorIconWithTextView two = (ChangeColorIconWithTextView)
        findViewById(R.id.id_indicator_two);
    ChangeColorIconWithTextView three = (ChangeColorIconWithTextView)
        findViewById(R.id.id_indicator_three);
```




```
mFragmentIndicator.add(one);
mFragmentIndicator.add(two);
mFragmentIndicator.add(three);

one.setOnClickListener(this);
two.setOnClickListener(this);
three.setOnClickListener(this);

one.setIconAlpha(1.0f);           //设置透明度 alpha,默认第一个
}
```

设置指示器单击事件监听。

```
public void onClick(View v) {
    int position = v.getId();
    resetOtherTabs(position);    //只要单击了某个 tab,其他的 tab 的透明度就设置为 0

    switch (position) {
        case R.id.id_indicator_one:
            mFragmentIndicator.get(0).setIconAlpha(1.0f);
            mViewPager.setCurrentItem(0, false);
            break;
        case R.id.id_indicator_two:
            mFragmentIndicator.get(1).setIconAlpha(1.0f);
            mViewPager.setCurrentItem(1, false);
            break;
        case R.id.id_indicator_three:
            mFragmentIndicator.get(2).setIconAlpha(1.0f);
            mViewPager.setCurrentItem(2, false);
            break;
    }
}
```

为 ViewPager 设置适配器,并设置页面滑动监听。

```
mViewPager.setAdapter(mAdapter);           //为 mViewPager 设置适配器
mViewPager.setOnPageChangeListener(this);  //为 mViewPager 设置页面滑动监听器
```

② Fragment 业务逻辑实现。

获取已建立的连接实例。

```
mActivity = (MainActivity) getActivity(); //获取当前 Activity
wRTConnect = mActivity.getWSNRConnect();  //获取 wRTConnect 实例
```

设置实时数据连接监听。

```
mApplication = (ZApplication) getActivity().getApplication(); //获取当前 Application
mApplication.registerOnIONWSNDataListener(InfraredFragment.this); //注册监听
当该 Fragment 可见时,发送查询命令。
//在 Fragment 隐藏和可见时都会调用
public void setUserVisibleHint(boolean isVisibleToUser) {
    //TODO Auto-generated method stub
    super.setUserVisibleHint(isVisibleToUser);
    if (isVisibleToUser) { //Fragment 可见时发送查询命令
        wRTConnect.sendMessage(mMac, "{A0=?}".getBytes());
    }
}
```



```
}
}
```

解析收到的实时数据，并根据 A0 值行人检测显示状态。

//传感器数据监听函数

```
public void onMessageArrive(String mac, String tag, String val) {
    //TODO Auto-generated method stub
    if (mac.equalsIgnoreCase(mMac)) { //检测 MAC 地址是否匹配
        if (tag.equals("A0")) {
            int v = Integer.parseInt(val);
            if (v == 1) {
                gifMV.setVisibility(View.VISIBLE); //显示行人
            } else {
                gifMV.setVisibility(View.GONE); //不显示行人
            }
        }
    }
}
```

③ 程序退出，释放智云实时数据连接。

```
public void onDestroyView() {
    //TODO Auto-generated method stub
    mApplication.unregisterOnIONWSNDataListener(this);
    super.onDestroy();
}
```

3. Web 端应用设计

根据 Web 应用编程接口定义，智能安防系统的应用设计主要采用实时数据 API 接口，JS 部分控制采集代码如下。

```
<script type="text/javascript">
var myZCloudID = "12345678"; //序列号
var myZCloudKey = "12345678"; //密钥

var mySensorMac1 = "00:12:4B:00:02:37:7E:7A"; //传感器的 MAC 地址（可燃气体传感器）
var mySensorMac2 = "00:12:4B:00:02:63:3C:B7"; //火焰传感器的 MAC 地址（人体红外）
var mySensorMac3 = "00:12:4B:00:02:63:3C:B7"; //人体红外的 MAC 地址
var mySensorMac4 = "00:12:4B:00:02:60:E3:A9"; //风扇传感器的 MAC 地址
var mySensorMac5 = "00:12:4B:00:02:63:3C:CF"; //声光报警传感器的 MAC 地址

var CombustibleGasStatus; //可燃气体检测状态
var FireStatus; //火焰状态
var RtStatus; //人体红外状态
var FanStatus; //风扇状态
var AlarmStatus; //声光报警状态

var COMFLAG = 0; //可燃气体是否允许检测标识
var FIREFLAG = 0; //火焰是否允许检测标识
var RTFLAG = 0; //人体红外是否允许检测标识

var rtc = new WSNRTConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
rtc.connect(); //数据推送服务连接
$("#ConnectState").text("数据服务连接中...");
```



```

rtc.onConnect = function () { //连接成功回调函数
    rtc.sendMessage(mySensorMac1, "{CD0=1,D0=?,A0=?}"); //向燃气传感器发送数据
    rtc.sendMessage(mySensorMac2, "{CD0=1,D0=?,A0=?}"); //向火焰传感器发送数据
    rtc.sendMessage(mySensorMac3, "{CD0=1,D0=?,A0=?}"); //向人体红外传感器发送数据
    rtc.sendMessage(mySensorMac4, "{D1=?}"); //向风扇传感器发送数据
    rtc.sendMessage(mySensorMac5, "{D1=?}"); //向声光报警传感器发送数据

    $("#ConnectState").text("数据服务连接成功!");
};

rtc.onConnectLost = function () { //数据服务掉线回调函数
    $("#ConnectState").text("数据服务掉线!");
};

rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
    // console.log(mac, " >>> ", dat);
    if (mac != mySensorMac1 && mac != mySensorMac2 && mac != mySensorMac3
        && mac != mySensorMac4 && mac != mySensorMac5) { //判断传感器 MAC 地址
        console.log("'" + mac + " not in sensors");
        return;
    }

    if (mac == mySensorMac1) { //可燃气体传感器
        if (dat[0] == '{' && dat[dat.length - 1] == '}') { //判断字符串首尾是否为{}
            subdat = dat.substr(1, dat.length - 2); //截取{}内的字符串
            its = subdat.split(','); //以','来分割字符串
            for (x in its) {
                t = its[x].split('='); //以'='来分割字符串
                if (t.length != 2) continue;
                if (t[0] == "D0"){
                    anNiu = document.getElementById("comctrlbutton");
                    D0 = parseInt(t[1]);
                    if (D0 == 1){
                        COMFLAG = 1;
                        anNiu.src = ("images/an-on.png");
                    }else{
                        COMFLAG = 0;
                        anNiu.src = ("images/an-off.png");
                        $("#ranqi").text("禁止检测!"); //显示状态
                        bG = document.getElementById("rq");
                        bG.src = ("images/ranqi-off.png");
                    }
                }
            }
            if (COMFLAG == 1) {
                if (t[0] == "A0") { //判断参数 A0
                    CombustibleGasStatus = parseInt(t[1]);
                    bG = document.getElementById("rq");
                    if (CombustibleGasStatus > 12) {
                        $("#ranqi").text("燃气浓度超标!"); //显示状态
                    }
                }
            }
        }
    }
}

```




```

        bG.src = ("images/ranqi-on.gif");
    } else {
        $("#ranqi").text("燃气浓度未超标!"); //显示状态
        bG.src = ("images/ranqi-off.png");
    }
}
}
}
}

if (mac == mySensorMac2) { //火焰传感器
    if (dat[0] == '{' && dat[dat.length - 1] == '}') { //判断字符串首尾是否为{}
        subdat = dat.substr(1, dat.length - 2); //截取{}内的字符串
        its = subdat.split(','); //以','来分割字符串
        for (.x in its) {
            t = its[x].split('='); //以'='来分割字符串
            if (t.length != 2) continue;
            if (t[0] == "D0"){
                anNiu = document.getElementById("firectrlbutton");
                D0 = parseInt(t[1]);
                if (D0 == 1){
                    FIREFLAG = 1;
                    anNiu.src = ("images/an-on.png");
                }else{
                    FIREFLAG = 0;
                    anNiu.src = ("images/an-off.png");
                    $("#huoyan").text("禁止检测!"); //显示状态
                    bG = document.getElementById("hy");
                    bG.src = ("images/hy-off.png");
                }
            }
        }
        if (FIREFLAG == 1){
            if (t[0] == "A0") { //判断参数 D1
                FireStatus = parseInt(t[1]);
                bG = document.getElementById("hy");
                if (FireStatus) {
                    $("#huoyan").text("检测到火焰!"); //显示状态
                    bG.src = ("images/hy-on.gif");
                } else {
                    $("#huoyan").text("未检测到火焰!"); //显示状态
                    bG.src = ("images/hy-off.png");
                }
            }
        }
    }
}

if (mac == mySensorMac3) { //人体红外传感器

```



```

    if (dat[0] == '{' && dat[dat.length - 1] == '}') { //判断字符串首尾是
否为{}
        subdat = dat.substr(1, dat.length - 2); //截取{}内的字符串
        its = subdat.split(','); //以','来分割字符串
        for (x in its) {
            t = its[x].split('='); //以'='来分割字符串
            if (t.length != 2) continue;
            if (t[0] == "D0"){
                anNiu = document.getElementById("rtctrlbutton");
                D0 = parseInt(t[1]);
                if (D0 == 1){
                    RTFLAG = 1;
                    anNiu.src = ("images/an-on.png");
                }else{
                    RTFLAG = 0;
                    anNiu.src = ("images/an-off.png");
                    $("#rthw").text("禁止检测!"); //显示状态
                    bG = document.getElementById("rt");
                    bG.src = ("images/rthw-off.png");
                }
            }
            if (RTFLAG == 1){
                if (t[0] == "A0") { //判断参数 D1
                    RtStatus = parseInt(t[1]);
                    bG = document.getElementById("rt");
                    if (RtStatus) {
                        $("#rthw").text("检测到人体!"); //显示状态
                        bG.src = ("images/rthw-on.gif");
                    } else {
                        $("#rthw").text("未检测到人体!"); //显示状态
                        bG.src = ("images/rthw-off.png");
                    }
                }
            }
        }
    }
}

if (mac == mySensorMac4) { //风扇
if (dat[0] == '{' && dat[dat.length - 1] == '}') { //判断字符串首尾是否为{}
    subdat = dat.substr(1, dat.length - 2); //截取{}内的字符串
    its = subdat.split(','); //以','来分割字符串
    for (x in its) {
        t = its[x].split('='); //以'='来分割字符串
        if (t.length != 2) continue;
        if (t[0] == "D1") { //判断参数 D1
            FanStatus = parseInt(t[1]);
            anNiu = document.getElementById("button04");
            bG = document.getElementById("fs");
            if (FanStatus) {

```



```

        anNiu.src = ("images/an-on.png");
        bG.src = ("images/fs-on.gif");
    } else {
        anNiu.src = ("images/an-off.png");
        bG.src = ("images/fs-off.png");
    }
}
}
}

if (mac == mySensorMac5) { //声光报警
    if (dat[0] == '{' && dat[dat.length - 1] == '}') { //判断字符串首尾是
否为{}

        subdat = dat.substr(1, dat.length - 2); //截取{}内的字符串
        its = subdat.split(','); //以','来分割字符串
        for (x in its) {
            t = its[x].split('='); //以'='来分割字符串
            if (t.length != 2) continue;
            if (t[0] == "D1") { //判断参数 D1
                AlarmStatus = parseInt(t[1]);
                anNiu = document.getElementById("button05");
                bG = document.getElementById("sgbj");
                if (AlarmStatus) {
                    anNiu.src = ("images/an-on.png");
                    bG.src = ("images/sgbj-on.gif");
                } else {
                    anNiu.src = ("images/an-off.png");
                    bG.src = ("images/sgbj-off.png");
                }
            }
        }
    }
}
}

};

function anniu04() {
    if (FanStatus) { //开启
        rtc.sendMessage(mySensorMac4, "{CD1=1,D1=?}"); //向风扇传感器发送数据(关)
    } else { //关闭
        rtc.sendMessage(mySensorMac4, "{OD1=1,D1=?}"); //向风扇传感器发送数据(开)
    }
}

function anniu05() {
    if (AlarmStatus) {
        rtc.sendMessage(mySensorMac5, "{CD1=1,D1=?}"); //向声光报警传感器发送数据(关)
    } else {
        rtc.sendMessage(mySensorMac5, "{OD1=1,D1=?}"); //向声光报警传感器发送数据(开)
    }
}
}

```




```
function comctrl(){
    if (COMFLAG == 0){
        rtc.sendMessage(mySensorMac1, "{OD0=1,D0=?,A0=?}");//修改报警状态值
    }else{
        rtc.sendMessage(mySensorMac1, "{CD0=1,D0=?,A0=?}");//修改报警状态值
    }
}

function firectrl(){
    if (FIREFLAG == 0){
        rtc.sendMessage(mySensorMac2, "{OD0=1,D0=?,A0=?}");//修改报警状态值
    }else{
        rtc.sendMessage(mySensorMac2, "{CD0=1,D0=?,A0=?}");//修改报警状态值
    }
}

function rtctrl(){
    if (RTFLAG == 0){
        rtc.sendMessage(mySensorMac3, "{OD0=1,D0=?,A0=?}");//修改报警状态值
    }else{
        rtc.sendMessage(mySensorMac3, "{CD0=1,D0=?,A0=?}");//修改报警状态值
    }
}
</script>
```

4.5.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个温湿度传感器无线节点, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到“C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples”文件夹下。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 SmartAlarm 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入智能灯光控制模块界面, 在主界面弹出“连接网关成功”消息后即表示链接到智云服务中心, 默认进入检测报警界面, 可以单击开关控制报警器, 或者开启燃气监测和火焰监测等, 如图 4.40 所示。



图 4.40 检测报警界面

滑动或者单击切换至风扇控制界面，可以单击开关控制风扇，如图 4.41 所示。



图 4.41 控制风扇界面

滑动或者单击切换至人体红外界面，当检测到有人时，画面中会动画显示人在走动，如图 4.42 所示。



图 4.42 人体红外界面

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 电脑接入到互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“SmartAlarm-Web\SmartAlarm”。



html”，进入智能灯光控制模块界面，在主界面右上角显示“数据服务连接成功！”消息后即表示链接到智云服务中心，在左侧单击不同传感器，会显示相应的状态或者开关按钮，如图 4.43 所示。



图 4.43 安防监控系统 Web 实现

4.5.6 总结与拓展

本任务应用多个传感器，开发了一个小型的智能安防综合应用系统，实现了数据监测和传感器实时控制等功能。开发者可以根据需要进行适当修改或者添加其他功能，使系统更加完善，提升编程能力。

4.6 任务 24：实验室管理系统开发（案例 12）

4.6.1 学习目标

- 掌握多种传感器的联合使用；
- 学会实验室管理系统项目开发和调试。

4.6.2 开发环境

硬件：可燃气体传感器 1 个，温湿度传感器 1 个，空气质量传感器 1 个，声光报警传感器 1 个，继电器 1 个，步进电机传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

4.6.3 原理学习

1. 系统设计目标

实验室智能管理系统中，开发者通过调用 ZCloud SDK API 中接口来实现实验室温湿度检测、空气质量数据检测、可燃气体浓度检测与可燃气体浓度过高时自动报警、灯光（继电器）控制、步进电机控制等功能操作。可以归为三个模块：实时数据显示模块、联动控制模块、

执行控制模块，系统设计功能及目标如图 4.44 所示。

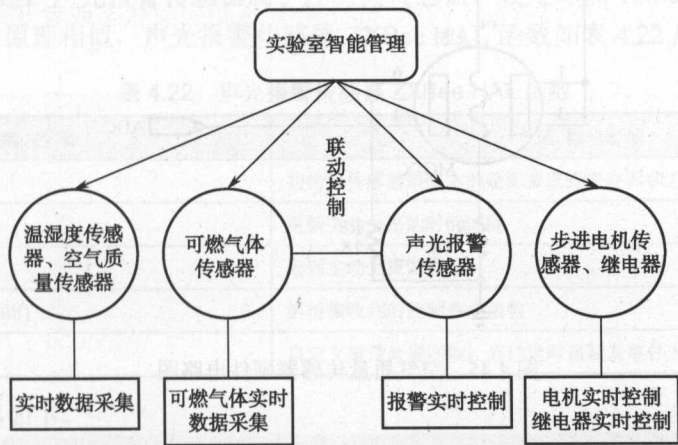


图 4.44 实验室智能管理系统功能模块

- (1) 实时数据显示模块：实时的显示实验室温湿度、空气质量 AQI 值。
- (2) 联动控制模块：可燃气体浓度值实时显示，当检测到可燃气体浓度过高时，自动控制进行声光报警。
- (3) 执行控制模块：在 Android 端进行单击事件来控制报警开闭、灯光开闭（灯光开闭是通过继电器来实现的）和步进电机转停。

2. 业务流程分析

实验室智能管理系统的通信流程可参考远程温湿度计这一章节，只是协调器组网时多添加了几个传感器而已。

3. 硬件原理

本任务中采用教学传感器：可燃气体传感器、温湿度传感器、空气质量传感器、声光报警传感器、继电器、步进电机传感器。

- (1) 可燃气体传感器、温湿度传感器、声光报警传感器、继电器和步进电机的工作原理参考前面章节内容。
- (2) 空气质量传感器硬件原理：本任务采用 MQ135 传感器来开发，MQ135 空气质量传感器属于敏感元件，通过微型 Al₂O₃ 陶瓷管、电导率低的 SnO₂ 敏感层，测量电极和加热器构成，固定在不锈钢或塑料制成的腔体内。传感器的工作条件由加热器来完成。封装好的器件有 6 只针状引脚，其中 4 个用于信号输出，另 2 个提供加热电压。当传感器处在存在有害气体的环境中时，其电导率就会随空气中有害气体浓度的增加而增加。MQ135 传感器在较大气体浓度范围内对有害气体有着较好的灵敏度，对氨气、硫化物、苯系蒸汽的灵敏度高，对烟雾和其他有害的监测也很好，可检测多种有害气体。通过电路设计即可将电导率的变化转换为与该气体浓度相对应的输出信号。

图 4.45 中的 ADC 口连接到 CC2530 的 P0_1 口，CC2530 是通过 ADC 来读取空气质量传感器的输出的值，当检测到空气质量有变化时，ADC 转换的值会发生变化。

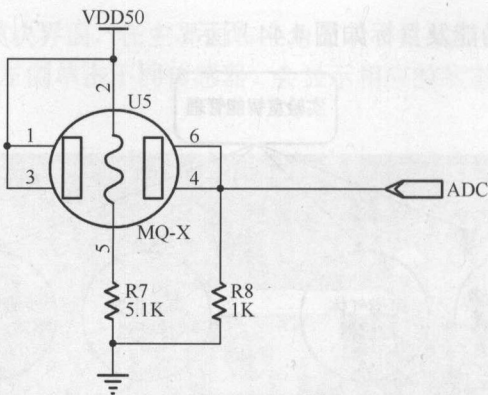


图 4.45 空气质量传感器硬件电路图

4.6.4 开发内容

1. 硬件层驱动设计

(1) ZXBee 智云数据通信协议。实验室智能管理中传感器的通信协议定义如表 4.21 所示。

表 4.21 相关传感器智云通信协议定义

传感器	属性	参数	权限	说 明
可燃气体/空气质量	数值	A0	R	可燃气体浓度值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔
温湿度	温度值	A0	R	温度值，浮点型：0.1 精度
	湿度值	A1	R	湿度值，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔
声光报警	电源开关	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示电源状态，OD1 为开、CD1 为关
步进电机	转动状态	D1(OD1/CD1)	R(W)	D1 的 Bit0 表示转动状态 OD1=3 为正转、CD1=1 为关
继电器	电源开关	D1(OD1/CD1)	R(W)	D1 的 Bit()表示电源状态，OD1 为开、CD1 为关

(2) 传感器驱动程序开发。可燃气体传感器、空气质量传感器、步进电机传感器、温湿度传感器、继电器的驱动开发参考前面章节，声光报警传感器的驱动开发流程如图 4.46 所示。

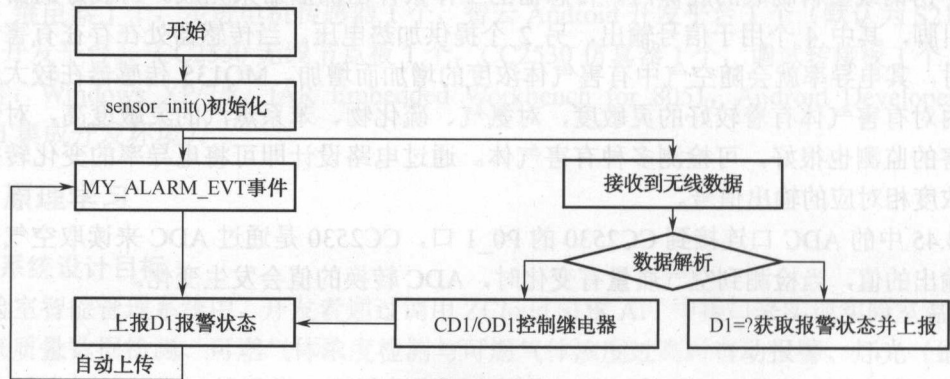


图 4.46 声光报警传感器驱动开发流程

根据 2.1 节所述，声光报警传感器属于控制类传感器，设定每隔 120 s 主动上传传感器数值，与继电器工作原理相似，声光报警传感器 ZXBee HAL 函数如表 4.22 所示。

表 4.22 声光报警传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	初始化传感器最基本的是配置选择寄存器和方向寄存器
updateV0()	更新主动上报的时间间隔
sensor_update()	处理主动上报的数据
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	自定义事件处理函数，启动定时器触发事件 MY_ALARM_EVT

部分程序代码如下。

```

/ *****宏定义*****/
#define SENSOR_RED      P0_1
#define SENSOR_GREEN    P1_3
#define SENSOR_BLUE     P0_6
#define SENSOR_BUZZER   P0_5
#define SENSOR_SEL      P0SEL
#define SENSOR_DIR      P0DIR
#define CLKDIV ( CLKCONCMD & 0x07 )

/ *****全局变量*****/
static uint8 D0 = 1;           //默认打开主动上报功能
static uint8 D1 = 0;           //Bit0 表示声光报警状态,1 报警, 0 不报警
static uint16 V0 = 120;        //V0 设置为上报时间间隔, 默认为 120 s
static uint16 myReportInterval = 120; //上报时间间隔, 单位为 s
static uint16 myAlarmDelay = 250; //报警灯呼吸灯延时, 毫秒
static uint8 Flag = 0;

/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
void sensor_init(void)
{
    //初始化传感器代码
    P1SEL &= ~ BV(3);
    P1DIR |= BV(3);

    SENSOR_SEL &= ~(BV(1) | BV(5) | BV(6));
    P0DIR |= BV(1) | BV(5) | BV(6);

    SENSOR_RED = 1;           //RGB_LDE 中的红色,1 灭,0 亮
    SENSOR_GREEN = 1;         //RGB_LDE 中的绿色,1 灭,0 亮
    SENSOR_BLUE = 1;          //RGB_LDE 中的蓝色,1 灭,0 亮
    SENSOR_BUZZER = 0;        //蜂鸣器, 1 开, 0 关

    //启动定时器, 触发事件: MY_REPORT_EVT、MY_OFF_EVT

```




```
osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10)
                                                    *1000));
osal_start_timerEx(sapi_TaskID, MY_ALARM_EVT, myAlarmDelay); // 开启报警灯循
环呼吸灯
}

/*****
*名称: updateV0()
*功能: 更新 V0 的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的 V0 值
*****/
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}

/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){ //若控制编码上报允许, 则 pData 的数据包中添加控制编码数据
        len = sprintf((char*)p, "D1=%u", D1);
        p += len;
        *p++ = ',';
    }

    //将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
    if (p - pData > 1) {
        pData[0] = '{';
        p[0] = 0;
        p[-1] = '}';

        zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                           AF_DEFAULT_RADIUS );
        HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
    }
}
```



```

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*      *pval -- 控制命令参数
*      *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest() 发送给协调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{
    int val;
    int ret = 0;

    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);

    //控制命令解析
    if (0 == strcmp("CD0", ptag)) {
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)) {
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D0=%u", D0);
        }
    }
    if (0 == strcmp("CD1", ptag)) {
        D1 &= ~val;
    }
    if (0 == strcmp("OD1", ptag)) {
        D1 |= val;
    }
    if (0 == strcmp("D1", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "D1=%u", D1);
        }
    }
    if (0 == strcmp("V0", ptag)) {
        if (0 == strcmp("?", pval)) {
            ret = sprintf(pout, "V0=%u", V0);
        }
        else{
            updateV0(pval);
        }
    }

    return ret;
}

```



```

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件 MY_REPORT_EVT
        osal_start_timerEx( sapi_TaskID, MY_REPORT_EVT, (uint16) (myReportInterval
                                                                    *1000));
    }

    //报警灯正常态呼吸灯, 绿灯每隔 1 s 闪烁, 蜂鸣器不响
    if ( event & MY_ALARM_EVT )
    {
        if(!D1) {
            SENSOR_RED = 1;
            SENSOR_GREEN = Flag;
            SENSOR_BLUE = 1;
            SENSOR_BUZZER = 0;
        }else {
            SENSOR_RED = Flag;
            SENSOR_GREEN = 1;
            SENSOR_BLUE = 1;
            SENSOR_BUZZER = Flag;
        }
        Flag = !Flag;
        osal_start_timerEx( sapi_TaskID, MY_ALARM_EVT, 3 *!D1 *myAlarmDelay +
                                                                    myAlarmDelay);
    }
}

```

2. 移动端应用设计

(1) 工程框架介绍。实验室智能管理系统工程框架如表 4.23 所示。

表 4.23 实验室智能管理系统工程框架介绍

包名 (类名)	说 明
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象, 定义应用程序全局单例对象
com.zonesion.tool 工具包	
ChangeColorIconWithTextView	自定义 View 的实现
com.zonesion.ui 子模块包	
AlarmFragment.java	燃气报警联动控制模块
ControlFragment.java	灯光电机控制模块



续表

包名（类名）	说 明
TemHumAndAirFragment	实时数据查询模块
com.zonesion.activity activity 包	
MainActivity.java	主 Activity

(2) 程序业务流程分析。根据 2.1 节智云 Android 应用编程接口定义，实验室智能管理系统的应用设计程序流程如图 4.47 所示。

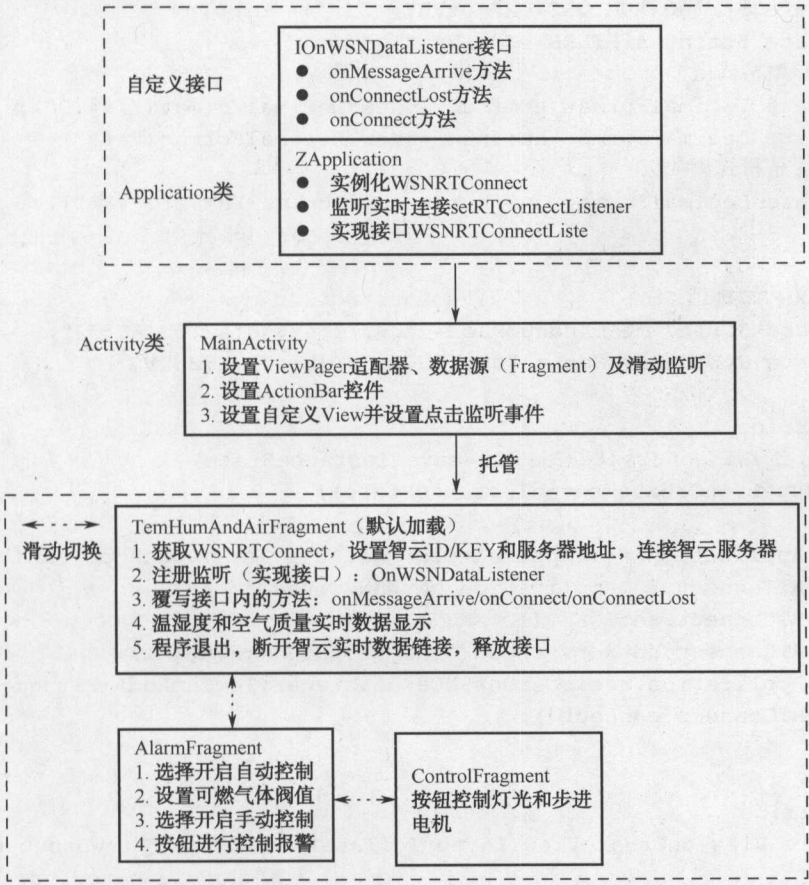


图 4.47 Android 应用程序流程

(3) 程序代码分析。接口定义、Application 类和 Activity 类的相关代码参考 4.2 节智慧窗帘控制系统，Fragment 相关代码如下。

① TemHumAndAirFragment 建立连接、注册监听，消息到来时进行数据分析并实时数据显示、断开连接。

```
public class TemHumAndAirFragment extends Fragment implements IONWSNDataListener
{
    private WSNRTConnect wRTConnect;
    private ZApplication mApplication;
```



```
public static String ID = "12345678";           //ID
public static String KEY = "12345678";         //密钥

private TextView temView;
private TextView humView;
private TextView airView;

//设置初始温度值
private String TEMTURE = "0.0℃";
//设置初始湿度值
private String HUMTURE = " 0%RH";
private String AIRTURE = " 0ppm";
//温湿度格式化
private DecimalFormat temfnum = new DecimalFormat("###.0");
private DecimalFormat humfnum = new DecimalFormat("###");
//AQI 值格式化
private DecimalFormat aqifnum = new DecimalFormat("###");

//定义 MAC 地址
private String mTemandhumMac = "00:12:4B:00:02:CB:A8:52";
private String airMac = "00:12:4B:00:02:63:3E:B5";

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mApplication = (ZApplication) getActivity().getApplication();
    wRTConnect = mApplication.getWSNRConnect();
    wRTConnect.setIdKey(ID, KEY);
    wRTConnect.setServerAddr("zhiyun360.com:28081");
    mApplication.registerOnWSNDataListener(TemHumAndAirFragment.this);
    wRTConnect.connect();
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.temhumair, container, false);

    temView = (TextView) v.findViewById(R.id.tem);
    humView = (TextView) v.findViewById(R.id.hum);
    airView = (TextView) v.findViewById(R.id.air);
    temView.setText(TEMTURE);
    humView.setText(HUMTURE);
    airView.setText(AIRTURE);

    return v;
}
```



```

@Override
public void onMessageArrive(String mac, String tag, String val) {

    if (mTemandhumMac.equalsIgnoreCase(mac)) { //判断 MAC 地址
        if (tag.equals("A0")) { //判断参数
            float v = Float.parseFloat(val);
            TEMTURE = "温度值: " + temfnum.format(v) + "°C";
            temView.setText(TEMTURE);
        }
        if (mTemandhumMac.equals(mac)) {
            if (tag.equals("A1")) { //判断参数
                float n = Float.parseFloat(val);
                HUMTURE = "湿度值: " + humfnum.format(n) + "%RH";
                humView.setText(HUMTURE);
            }
        }
    }
    if (mac.equalsIgnoreCase(airMac)) {
        if (tag.equals("A0")) {
            float v = Float.parseFloat(val);
            AIRTURE = "AQI: " + aqifnum.format(v) + "ppm";
            airView.setText(AIRTURE);
        }
    }
}

@Override
public void onConnect() {
    //TODO Auto-generated method stub
    wRTConnect.sendMessage(mTemandhumMac, "{A0=?,A1=?}".getBytes());
    wRTConnect.sendMessage(airMac, "{A0=?}".getBytes());
}

@Override
public void onConnectLost() {
    //TODO Auto-generated method stub
}

public void onDestory() {
    super.onDestroy();
    wRTConnect.disconnect();
    mApplication.unregisterOnWSNDataListener(TemHumAndAirFragment.this);
}
}

```

② CombustibleFragment 联动控制报警。

```

controlBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        if (FLAG == 0) { //关闭状态

```




```

        command = "{OD1=1,D1=?}";
    } else {
        //打开状态
        command = "{CD1=1,D1=?}";
    }
    //发送查询状态的命令
    WRTConnect.sendMessage(mMac, command.getBytes());
}
});

```

Android 客户端有两种控制模式：自动控制模式和手动控制模式。选择开启自动控制，设置阈值，当可燃气体浓度达到指定的阈值时进行报警；选择开启手动控制，可以单击按钮进行报警控制。

控制模式选择。

//自动控制

```

autoButton.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        autoButton.setBackgroundResource(R.drawable.autobgpress);
        controlButton.setBackgroundResource(R.drawable.buttonbg);
        controlFlag = 1;
        //自动控制状态
        autoCombustibleValue.setVisibility(View.VISIBLE);
        combustibleSb.setVisibility(View.VISIBLE);
        controlBtn.setVisibility(View.INVISIBLE);
    }
});

```

//手动控制

```

controlButton.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        autoButton.setBackgroundResource(R.drawable.autobg);
        controlButton.setBackgroundResource(R.drawable.buttonbgpress);
        controlFlag = 0;
        //自动控制状态
        autoCombustibleValue.setVisibility(View.INVISIBLE);
        combustibleSb.setVisibility(View.INVISIBLE);
        controlBtn.setVisibility(View.VISIBLE);
    }
});

```

自动控制模式。

```

if (controlFlag == 1) {
    //开启了自动控制
    if (v >= value) {
        //超出了阈值，发送打开报警器的命令
        command = "{OD1=1,D1=?}";
        WRTConnect.sendMessage(mAlarmMac, command.getBytes());
    } else {
        command = "{CD1=1,D1=?}";
        WRTConnect.sendMessage(mAlarmMac, command.getBytes());
    }
}

```



```

    }
}

```

手动控制模式。

```

//获取 ImageButton 并设置监听
controlBtn = (ImageButton) v.findViewById(R.id.controlBtn);
controlBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        if (alarmFlag == 0) {                                //关闭状态
            command = "{OD1=1,D1=?}";
        } else {                                            //打开状态
            command = "{CD1=1,D1=?}";
        }
        //发送命令
        wRTConnect.sendMessage(mAlarmMac, command.getBytes());
    }
});
if (alarmFlag == 1) {
    controlBtn.setBackgroundResource(R.drawable.bgbuttonon);
    imageView.setBackgroundResource(R.drawable.alarmon);
} else {
    controlBtn.setBackgroundResource(R.drawable.bgbuttonoff);
    imageView.setBackgroundResource(R.drawable.alarmoff);
}
}

```

③ ControlFragment, 按钮控制灯光开闭。

```

changeBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //TODO Auto-generated method stub
        if (flag.equals("on")) {
            //关闭
            command = "{CD1=1,D1=?}";
        } else if (flag.equals("off")) {
            //打开
            command = "{OD1=1,D1=?}";
        }
        wRTConnect.sendMessage(lightMac, command.getBytes());
    }
});

```

按钮控制电机转停。

```

stepButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        if (flag.equals("on")) {
            command = "{CD1=1,D1=?}";
        }
        else if (flag.equals("off")) {
            command = "{OD1=3,D1=?}";
        }
    }
});

```



```

    }
    WRTConnect.sendMessage(stepMac, command.getBytes());
  }
});

```

3. Web 端应用设计

根据 Web 应用编程接口定义, 实验室智能管理系统的应用设计主要采用实时数据 API 接口和控制 API 接口, JS 部分代码如下。

```

<!DOCTYPE html>
<html>

<head lang="en">
<meta charset="UTF-8">
<title>
实验室智能管理
</title>
<link rel="stylesheet" type="text/css" href="css/bootstrap.css" />
<script src="js/jquery-1.11.0.min.js" type="text/javascript">
</script>
<script src="js/bootstrap.min.js" type="text/javascript">
</script>
<script src="js/WSNRTConnect.js" type="text/javascript">
</script>
<style type="text/css">
    body {margin: 0;padding: 0;background-color: #a3cf9a;font-family: '微软
雅黑', '黑体', '宋体', serif;}
    .header {width: 100%;background-color: #89ae82;} h1 {padding: 0;margin:
0 0 15px;font-size: 24px;color: #fff;line-height: 3em;font-weight: 100;}
    h1 small { font-size: 12px;color: #fff;margin-left: 10px;} h1 span {float:
right;font-size:
14px;} .content {width: 1200px;margin: 0 auto;} .body-left {float: left;
width:
311px; margin-right:10px;text-align: center;color: #fff;} .body-left .
left-nav
{float: left;width: 180px;list-style: none;margin: 0;padding: 0;text-align:
left;} .body-left .left-nav .line01 {background: #b4ddac;} .body-left .
left-nav
.line02 {background: #9ecb96;} .body-left .left-nav .active {background:
#89ae82;} .body-left .left-nav a {position:relative;display:block;line-
height:54px;text-decoration:none;color:
#fff;padding-left:60px;} .body-left .left-nav a img {position: absolute;
top:
7px;left: 10px;} .body-left .content {float: left;width: 131px;background:
#89ae82;height: 378px;} .body-left .content .box {position: absolute;
margin:
20px 10px;width: 111px;} .body-left .content .box h3 {font-size: 16px;
padding:
0;margin: 20px 0;} .body-left .content .box p {padding: 0;margin: 10px
0;} .body-left .content .box p span {font-size: 36px;} .body-left .content
.active {z-index: 10;} #button {width: 80px;} .body-right {position:

```




```

relative;float:
    left;width: 879px;} .body-right .sgbj {position: absolute;top: 343px;
left:
    472px;width: 50px;} .body-right .deng {position: absolute;top: 2px;left:
    189px;} .body-right .chuanglian {position: absolute;top: 42px;left: 85px;}
    .body-right .hy {position: absolute;top: -26px;left: 2px;width: 100px;}
</style>
<script type="text/javascript">
    var AirQuality;
    var myZCloudID = "12345678";    //序列号
    var myZCloudKey = "12345678";    //密钥
    var mySensorMac1 = "00:12:4B:00:02:CB:A8:52"; //传感器的MAC 地址（温湿度传感器）
    var mySensorMac2 = "00:12:4B:00:02:63:3E:B5"; //传感器的MAC 地址（空气质量传感器）
    var mySensorMac3 = "00:12:4B:00:03:A7:E1:17"; //传感器的MAC 地址（继电器）
    var mySensorMac4 = "00:12:4B:00:02:63:3C:93"; //传感器的MAC 地址（步进电机）
    var mySensorMac5 = "00:12:4B:00:02:37:7E:7A"; //传感器的MAC 地址（可燃气体传感器）
    var mySensorMac6 = "00:12:4B:00:02:63:3C:CF"; //传感器的MAC 地址（声光报警）
    var rtc = new WSNRTConnect(myZCloudID, myZCloudKey); //创建数据连接服务对象
    rtc.connect(); //数据推送服务连接
    $("#ConnectState").text("数据服务连接中...");

    rtc.onConnect = function() { //连接成功回调函数
        rtc.sendMessage(mySensorMac2, "{A0=?}"); //向可燃气体传感器发送数据
        $("#ConnectState").text("数据服务连接成功!");
    };

    rtc.onConnectLost = function() { //数据服务掉线回调函数
        $("#ConnectState").text("数据服务掉线!");
    };

    rtc.onmessageArrive = function(mac, dat) { //消息处理回调函数
        //接收数据过滤（温湿度传感器）
        if ((mac == mySensorMac1) && (dat.indexOf(",") == -1)) {
            var aisle = dat.substring(dat.indexOf("{") + 1, dat.indexOf("="));
            if (aisle == "A0") {
                //将原始数据的数字部分分离出来
                dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf("}"));
                $("#temperature").text(dat); //显示温度值
            }
            if (aisle == "A1") {
                //将原始数据的数字部分分离出来
                dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf("}"));
                $("#humidity").text(dat); //显示湿度值
            }
        }
        //接收数据过滤（空气质量传感器）
        if ((mac == mySensorMac2) && (dat.indexOf(",") == -1)) {
            dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf("}"));
            AirQuality = dat;
        }
    }

```



```
//接收数据过滤(可燃气体传感器)
if ((mac == mySensorMac5) && (dat.indexOf(",") == -1)) {
    dat = dat.substring(dat.indexOf("=") + 1, dat.indexOf("}"));
    if (parseInt(dat) > 10) { //空气质量数值大于10 报警
        document.getElementById("hy").src = ("images/hy-bj.gif");
    }
}
};

var flag1 = true;
function anniu01() {
    var anNiu = document.getElementById("button01");
    if (flag1) {
        rtc.sendMessage(mySensorMac1, "{A0=?}"); //向温度传感器发送数据获取温度值
        anNiu.src = ("images/an-on.png");
    } else {
        $("#temperature").text(""); //关闭温度值显示
        anNiu.src = ("images/an-off.png");
    }
    flag1 = !flag1
}

var flag2 = true;
function anniu02() {
    var anNiu = document.getElementById("button02");
    if (flag2) {
        rtc.sendMessage(mySensorMac1, "{A1=?}"); //向湿度传感器发送数据获取湿度值
        anNiu.src = ("images/an-on.png");
    } else {
        $("#humidity").text(""); //关闭湿度值显示
        anNiu.src = ("images/an-off.png");
    }
    flag2 = !flag2
}

var flag3 = true;
function anniu03() {
    var anNiu = document.getElementById("button03");
    if (flag3) {
        rtc.sendMessage(mySensorMac2, "{A0=?}"); //向空气质量传感器发送数据
        $("#airQuality").text(AirQuality); //显示空气质量数值
        anNiu.src = ("images/an-on.png");
    } else {
        $("#airQuality").text(""); //关闭空气质量数值显示
        anNiu.src = ("images/an-off.png");
    }
    flag3 = !flag3
}

var flag4 = true;
function anniu04() {
    var anNiu = document.getElementById("button04");
    var bG = document.getElementById("deng");
```



```
if (flag4) {
    rtc.sendMessage(mySensorMac3, "{OD1=1,D1=?}"); //向继电器发送数据 (开)
    anNiu.src = ("images/an-on.png");
    bG.src = ("images/deng_on.png");
} else {
    rtc.sendMessage(mySensorMac3, "{CD1=1,D1=?}"); //向继电器发送数据 (关)
    anNiu.src = ("images/an-off.png");
    bG.src = ("images/deng_off.png");
}
flag4 = !flag4
}
var flag5 = true;
function anniu05() {
    var anNiu = document.getElementById("button05");
    var bG = document.getElementById("chuanglian");
    if (flag5) {
        rtc.sendMessage(mySensorMac4, "{OD1=3,D1=?}"); //向步进电机发送数据 (正转)
        anNiu.src = ("images/an-on.png");
        bG.src = ("images/chuanglian_on.gif");
    } else {
        rtc.sendMessage(mySensorMac4, "{CD1=1,D1=?}"); //向步进电机发送数据 (停止)
        anNiu.src = ("images/an-off.png");
        bG.src = ("images/chuanglian_off.gif");
    }
    flag5 = !flag5
}
var flag6 = true;
function anniu06() {
    var anNiu = document.getElementById("button06");
    var bG = document.getElementById("hy");
    if (flag6) {
        rtc.sendMessage(mySensorMac5, "{A0=?}"); //向可燃气体传感器发送数据
        anNiu.src = ("images/an-on.png");
        bG.src = ("images/hy-on.png");
    } else {
        anNiu.src = ("images/an-off.png");
        bG.src = ("images/hy-off.png");
    }
    flag6 = !flag6
}
var flag7 = true;
function anniu07() {
    var anNiu = document.getElementById("button07");
    var bG = document.getElementById("sgbj");
    if (flag7) {
        rtc.sendMessage(mySensorMac6, "{OD1=1,D1=?}"); //向声光报警传感器发送数据 (开)
        anNiu.src = ("images/an-on.png");
        bG.src = ("images/sgbj-on.gif");
    } else {
        rtc.sendMessage(mySensorMac6, "{CD1=1,D1=?}"); //向声光报警传感器发送数据 (关)
```




```
        anNiu.src = ("images/an-off.png");
        bG.src = ("images/sgbj-off.png");
    }
    flag7 = !flag7
}
</script>
</head>

<body>
<div class="header">
<div class="content">
<h1>
实验室智能管理
<small>
温湿度、空气质量、灯光、步进电机、燃气、声光报警
</small>
<span>
<lable id="ConnectState">
</lable>
</span>
</h1>
</div>
</div>
<div class="content">
<div class="body-left">

<div>
<ul class="left-nav">
<li class="line01 active">
<a href="#title1" data-toggle="tab">

温度计
</a>
</li>
<li class="line02">
<a href="#title2" data-toggle="tab">

湿度计
</a>
</li>
<li class="line01">
<a href="#title3" data-toggle="tab">

空气质量检测器
</a>
</li>
<li class="line02">
<a href="#title4" data-toggle="tab">

灯光
```



```

</a>
</li>
<li class="line01">
<a href="#title5" data-toggle="tab">

步进电机
</a>
</li>
<li class="line02">
<a href="#title6" data-toggle="tab">

可燃气体检测
</a>
</li>
<li class="line01">
<a href="#title7" data-toggle="tab">

声光报警器
</a>
</li>
</ul>
<div class="content">
<div class="box fade in active" id="title1">
<h3>
开始检查
</h3>

<p>
温度值
<br />
<span>
<lable id="temperature">
</lable>
</span>
</p>
</div>
<div class="box fade" id="title2">
<h3>
开始检查
</h3>

<p>
湿度值
<br />
<span>
<lable id="humidity">
</lable>
</span>
</p>
</div>

```



```
<div class="box fade" id="title3">
<h3>
开始检查
</h3>

<p>
        PM2.5
<br />
<span>
<lable id="airQuality">
</lable>
</span>
</p>
</div>
<div class="box fade" id="title4">
<h3>
开关
</h3>

</div>
<div class="box fade" id="title5">
<h3>
开关
</h3>

</div>
<div class="box fade" id="title6">
<h3>
开关
</h3>

</div>
<div class="box fade" id="title7">
<h3>
开关
</h3>

</div>
</div>
</div>
</div>
<div class="body-right">




<!-- 报警: hy-bj.png -->

</div>
</div>
```




</body>

</html>

4.6.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个可燃气体传感器无线节点、1 个温湿度传感器、一个空气质量传感器、一个声光报警传感器、一个继电器、一个步进电机传感器, 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到 “C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples” 文件夹下。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 SmartLab 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入系统默认加载的 Fragment (TemHumAndAirFragment), 进入实时数据采集显示界面, 在界面弹出 “连接网关成功” 消息后即表示连接到智云服务中心显示, 并会实时显示温湿度值、空气质量值, 如图 4.48 所示。



图 4.48 实时数据显示界面

(4) 滑动或单击切换至报警界面, 显示当前燃气值并可以选择手动或自动模式控制报警, 如图 4.49 所示。



图 4.49 控制报警器界面

(5) 滑动或单击切换至控制界面，可以单击开关控制灯光和步进电机，如图 4.50 所示。



图 4.50 控制传感器界面

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 将电脑接入到互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“SmartLab-Web\SmartLab.html”，进入实验室智能管理界面，在主界面右上角显示“数据服务连接成功！”消息后即表示连接到智云服务中心，在左侧有此次实验用到的传感器的开关控制，可以开启进行控制实时数据采集、灯光开、步进电机工作、报警等等。在右侧是整个实验室的图片显示，并直观地进行显示左侧单击进行的操作，如图 4.51 所示。



图 4.51 实验室智能管理系统 Web 端

4.6.6 总结与拓展

(1) 本任务用到了很多传感器,但传感器的联合使用却少,开发者可以发挥自己的想象力,用两个或多个传感器联合控制来实现实验室管理。

(2) 本任务中所构造的布局不是很好,由于布局适配问题,在大屏幕下,不能显示得很好。但是 Android 提供多布局,开发者可在 Eclipse 中在 SmartLab 的 res 文件下创建 layout-large 文件夹,在此文件夹下建立同名的 layout 布局,再自己编写布局即可,这样就能很好地适配不同的终端。

4.7 任务 25: 无线抄表系统开发 (案例 13)

4.7.1 学习目标

- 掌握用户数据接口的使用;
- 学会无线抄表系统项目开发和调试。

4.7.2 开发环境

硬件: 温湿度传感器 1 个, 模拟节点 (电表) 1 个, 智云 Android 开发平台 1 个 (默认为 S210 系列 Android 开发平台), CC2530 无线节点板 1 个, CC2530 仿真器 1 个, 调试转接板 1 个。

软件: Windows XP/7/8, IAR Embedded Workbench for 8051, Android Developer Tools (Android 集成开发环境)。

4.7.3 原理学习

1. 系统设计目标

在整个设计中,将模拟节点模拟出一个电表,并在 Android 端或 Web 端对采集到的数据进行显示。显然电表属于采集类传感器,由于没有硬件支持,使用随机数来产生电压和电流,



让模拟节点向上层应用传输由随机数产生的电压和电流，并在上层应用中对其中的功率和用电量用公式进行计算得出，无线抄表系统设计功能及目标如图 4.52 所示。

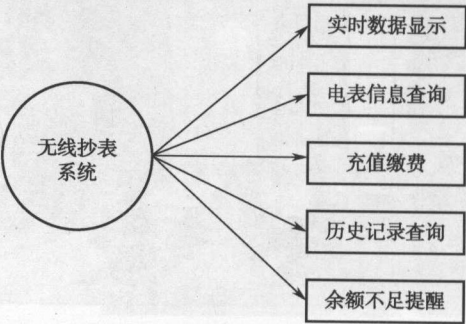


图 4.52 无线抄表系统功能模块

其中，实时数据显示模块采用的是实时数据服务；电表信息查询模块采用的是历史数据接口。用户余额采用的用户数据接口，智云 SDK 中，用户数据接口的使用方法介绍如下。

智云用户数据接口提供私有的数据库使用权限，实现多客户端间共享的私有数据进行存储、查询和使用。私有数据存储采用 Key-Value 型数据库服务，编程接口更简单高效。

基于 Android 的接口基本使用如表 4.24 所示（具体参考 ZCloudAPI 文档）。

表 4.24 接口函数

函 数	参 数 说 明	功 能
newWSNProperty(String myZCloudID,String myZCloudKey);	myZCloudID:智云账号 myZCloudKey:智云密钥	初始化用户数据对象,并初始化智云 ID 及密钥
put(String key,String value);	key:名称 value:内容	创建用户应用数据
get();	无	获取所有的键值对
get(String key);	key:名称	获取指定 Key 的 Value 值

2. 业务流程分析

无线抄表系统从传输过程分为三部分：传感节点、网关、客户端（Android 和 Web），通信流程图如图 4.53 所示，具体通信描述如下。

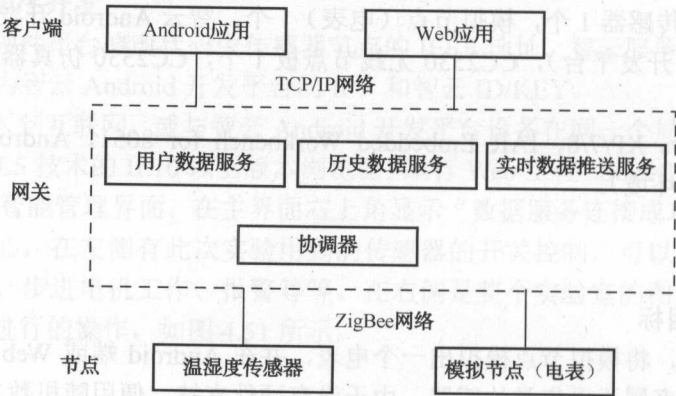


图 4.53 无线抄表业务流程

(1) 传感器节点通过 ZigBee 网络与网关的协调器进行组网，网关的协调器通过串口与网关进行数据通信。

(2) 底层节点的数据通过 ZigBee 网络将数据传送给协调器，协调器通过串口将数据转发给网关服务，通过实时数据推送服务将数据推送给所有连接网关的客户端；通过历史数据存储服务将数据存储到数据中心；通过用户数据服务从服务器获取数据或者将数据推送至服务器。

(3) Android 应用通过调用 ZCloud SDK API 的实时数据连接接口实现实时数据采集的功能，通过历史数据存储服务实现历史数据查询；通过调用用户数据接口实现数据的读写，详细实现参考 4.7.4 节开发内容代码实现部分。

3. 硬件原理

本任务，用到传感器分别有温湿度传感器、模拟节点，由于没有硬件支持，所以采用随机数产生来获取数据，进而模拟电表。

其中，温湿度传感器的硬件原理可参考前面章节（3.1.3 节中远程温湿度计硬件原理），模拟节点（电表）无硬件传感器、硬件原理为无。

4.7.4 开发内容

1. 硬件层驱动设计

温湿度传感器的驱动设计可参考 3.1 节远程温湿度计系统开发，另外模拟节点（电表）的驱动设计如下。

(1) ZXBee 智云数据通信协议，定义模拟节点（电表）的通信协议如表 4.25 所示。

表 4.25 相关传感器智云通信协议定义

传感器	属性	参数	权限	说 明
模拟节点 (电表)	电压	A0	R	电压值，浮点型：0.1 精度
	电流	A1	R	电流值，浮点型：0.1 精度
	功率	A2	R	功率值，浮点型：0.1 精度
	电量	A3	R	用电量，浮点型：0.1 精度
	上报状态	D0(OD0/CD0)	R(W)	D0 的 Bit0 表示电压上传状态、Bit1 表示电流上传状态
	上报间隔	V0	RW	修改主动上报的时间间隔

(2) 传感器驱动程序开发。模拟节点（电表）程序逻辑如图 4.54 所示。

根据 2.1 节所述，模拟节点（电表）属于定时采集类传感器，设定每隔 30 s 主动上报传感器数值。相关传感器 ZXBee HAL 函数如表 4.26 所示。

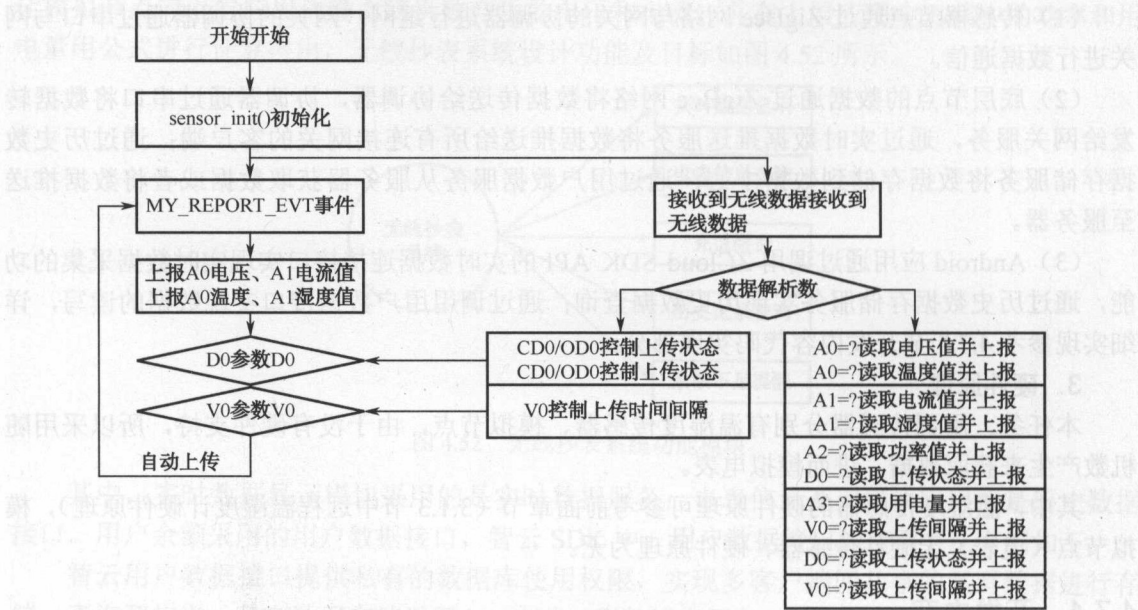


图 4.54 模拟节点（电表）程序逻辑

表 4.26 相关传感器 ZXBee HAL 函数

函数名称	函数说明
sensor_init()	初始化节点最基本的是配置选择寄存器和方向寄存器
updateV0()	更新主动上报的时间间隔
updateA0()/updateA1()	更新节点电压值、电流值
sensor_update()	上报采集到的数据
usr_process_command_call()	解析接收到的控制命令函数
MyEventProcess()	自定义事件处理函数，启动定时器触发事件 MY_REPORT_EVT

部分程序代码如下。

```

/ *****宏定义*****/
#define random(x) (rand()%x)
/ *****全局变量*****/
static uint8 D0 = 7; //默认打开主动上报功能
static float A0 = 0.0; //A0 存储电压值
static float A1 = 0.0; //A1 存储电流值
static float A2 = 0.0; //A2 存储功率值
static float A3 = 0.0; //A3 存储电量值
static uint16 V0 = 30; //V0 设置为上报时间间隔，默认为 30 s
static uint16 myReportInterval = 30; //上报时间间隔，单位为 s
/*****
*名称: sensor_init()
*功能: 传感器硬件初始化
*****/
```




```

void sensor_init(void)
{
    //启动定时器, 触发事件: MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand()%10)
                                                                *1000));

    osal_start_timerEx(sapi_TaskID, MY_A3REPORT_EVT, (uint16)(myReportInterval
                                                                *1000));
}
/*****
*名称: updateV0()
*功能: 更新 V0 的值
*参数: *val -- 待更新的变量
*返回: V0 -- 返回更新后的 V0 值
*****/
uint16 updateV0(char *val)
{
    //将字符串变量 val 解析转换为整型变量赋值
    myReportInterval = atoi(val);
    V0 = myReportInterval;

    return V0;
}
/*****
*名称: updateA0()
*功能: 更新 A0 的值
*参数: 无
*返回: A0 -- 返回更新后的 A0 值
*****/
float updateA0(void)
{
    A0 = 200 + random(49) / 1.23;                //随机生成电压值, 200~240 V 之间

    return A0;
}

/*****
*名称: updateA1()
*功能: 更新 A1 的值
*参数: 无
*返回: A1 -- 返回更新后的 A1 值
*****/
float updateA1(void)
{
    float temp = random(12) / 1.23;                //随机生成电流值, 0.5~10 A 之间
    if (temp < 0.5)
    {
        temp += 0.5;
    }
    A1 = temp;
}

```



```
        return A1;
    }
/*****
*名称: updateA2()
*功能: 更新 A2 的值
*参数: 无
*返回: A2 -- 返回更新后的 A2 值
*****/
float updateA2(void)
{
    A2 = A0 *A1;                //计算功率
    return A2;
}

/*****
*名称: updateA3()
*功能: 更新 A3 的值
*参数: 无
*返回: A3 -- 返回更新后的 A3 值
*****/
float updateA3(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    A3 = A2 *30 / 3600;          //每 30 s 计算一次用电量

    len = sprintf((char*)p, "A3=%.1f", A3);
    p += len;
    *p++ = ',';

    if (p - pData > 1) {
        pData[0] = '{';
        p[0] = 0;
        p[-1] = '}';

        zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                           AF_DEFAULT_RADIUS );
        HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK );    //通信 LED 闪烁一次
    }

    return A3;
}

/*****
*名称: sensor_update()
*功能: 处理主动上报的数据
*****/
```



```

void sensor_update(void)
{
    uint16 cmd = 0;
    uint8 pData[128];
    uint8 *p = pData + 1;
    int len;

    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){ //若电压上报允许, 则 pData 的数据包中添加温度数据
        updateA0();
        len = sprintf((char*)p, "A0=%.1f", A0);
        p += len;
        *p++ = ',';
    }
    if ((D0 & 0x02) == 0x02){ //若电流上报允许, 则 pData 的数据包中添加湿度数据
        updateA1();
        len = sprintf((char*)p, "A1=%.1f", A1);
        p += len;
        *p++ = ',';
    }
    if ((D0 & 0x04) == 0x04){ //若功率上报允许, 则 pData 的数据包中添加湿度数据
        updateA2();
        len = sprintf((char*)p, "A2=%.1f", A2);
        p += len;
        *p++ = ',';
    }

    //将需要上传的数据进行打包操作, 并通过 zb_SendDataRequest() 发送到协调器
    if (p - pData > 1) {
        pData[0] = '{';
        p[0] = 0;
        p[-1] = '}';

        zb_SendDataRequest( 0, cmd, p-pData, pData, 0, AF_ACK_REQUEST,
                                                                    AF_DEFAULT_RADIUS );
        HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK ); //通信 LED 闪烁一次
    }
}

/*****
*名称: usr_process_command_call()
*功能: 解析收到的控制命令
*参数: *ptag -- 控制命令名称
*       *pval -- 控制命令参数
*       *pout -- 控制响应数据, 将数据返回给上级调用, 通过 zb_SendDataRequest{} 发送给协调器
*返回: ret -- pout 字符串长度
*****/
int usr_process_command_call(char *ptag, char *pval, char *pout)
{

```




```
int val;
int ret = 0;

//将字符串变量 pval 解析转换为整型变量赋值
val = atoi(pval);

//控制命令解析
if (0 == strcmp("CD0", ptag)) {
    D0 &= ~val;
}
if (0 == strcmp("OD0", ptag)) {
    D0 |= val;
}
if (0 == strcmp("D0", ptag)) {
    if (0 == strcmp("?", pval)) {
        ret = sprintf(pout, "D0=%u", D0);
    }
}
if (0 == strcmp("A0", ptag)) {
    if (0 == strcmp("?", pval)) {
        updateA0();
        ret = sprintf(pout, "A0=%.1f", A0);
    }
}
if (0 == strcmp("A1", ptag)) {
    if (0 == strcmp("?", pval)) {
        updateA1();
        ret = sprintf(pout, "A1=%.1f", A1);
    }
}
if (0 == strcmp("A2", ptag)) {
    if (0 == strcmp("?", pval)) {
        updateA2();
        ret = sprintf(pout, "A2=%.1f", A2);
    }
}
if (0 == strcmp("V0", ptag)) {
    if (0 == strcmp("?", pval)) {
        ret = sprintf(pout, "V0=%u", V0);
    } else {
        updateV0(pval);
    }
}

return ret;
}

/*****
*名称: MyEventProcess()
*功能: 自定义事件处理
*****/
```

//更新电压数值

//更新电流数值

//更新功率数值

```
*参数: event -- 事件编号
*****/
void MyEventProcess( uint16 event )
{
    if (event & MY_REPORT_EVT) {
        sensor_update();
        //启动定时器, 触发事件: MY_REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16) (myReportInterval
                                                                    *1000));
    }
    if (event & MY_A3REPORT_EVT) {
        updateA3();
        //启动定时器, 触发事件: MY_A3REPORT_EVT
        osal_start_timerEx(sapi_TaskID, MY_A3REPORT_EVT, (uint16) (myReportInterval
                                                                    *1000));
    }
}
```

2. 移动端应用设计

(1) 工程框架介绍。无线抄表系统工程框架如表 4.27 所示。

表 4.27 无线抄表系统工程框架

包名（类名）	说 明
com.zonesion.app 应用包	
IONWSDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象, 定义应用程序全局单例对象
com.zonesion.tool 工具包	
ChangeColorIconWithTextView	自定义 View 的实现
MyDialog	自定义 ProgressDialog 的实现
com.zonesion.ui 子模块包	
InfoFragment.java	实时数据显示模块
HistoryChartFragment.java	历史记录查询模块
ChargeFragment.java	余额查询、缴费充值模块
com.zonesion.activity activity 包	
MainActivity.java	主 Activity

(2) 程序业务流程分析。无线抄表系统调用的是实时数据 API 接口、历史数据 API 接口和用户数据 API 接口，程序的实现流程如图 4.55 所示。

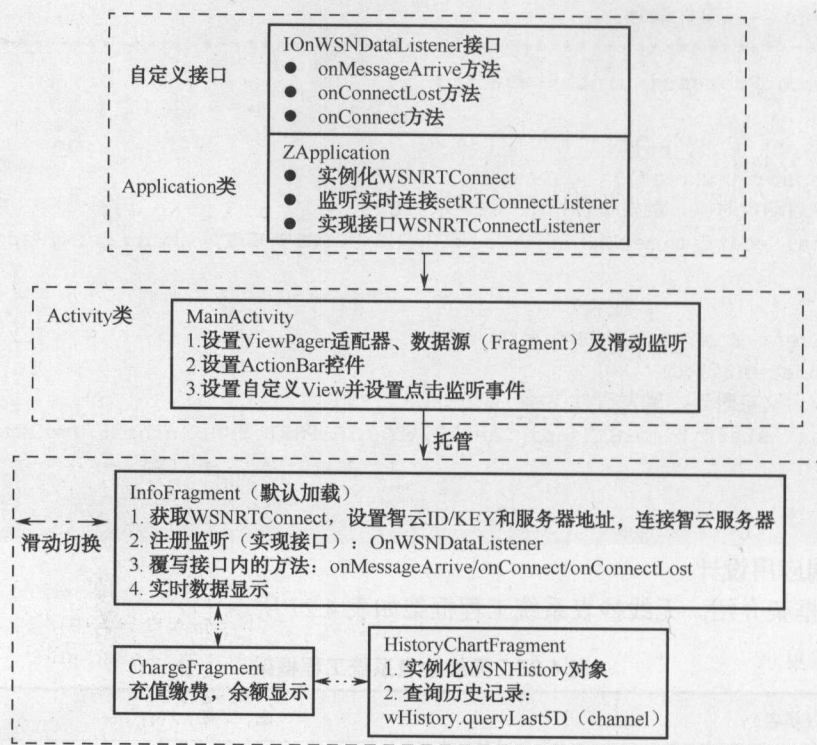


图 4.55 抄表系统程序设计逻辑

(3) 程序代码剖析。

① ZApplication 框架说明参考智能灯光控制这一章节。

② MainActivity 实现了 ViewPager 数据源设置, 各个 Fragment 模块加载, 以及自定义 View 的单击事件和 ActionBar 控件的应用, 核心源码如下所示。

```
public class MainActivity extends FragmentActivity implements
    OnPageChangeListener, OnClickListener {
    .....
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        .....
        setOverflowShowingAlways(); //显示 ActionBar 导航栏的 Overflow 按钮
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true); //使用程序图标作为 home icon
        actionBar.setDisplayHomeAsUpEnabled(true); //显示返回的箭头, 并可通过
        //onOptionsItemSelected() 进行监听, 其资源ID为Android.R.id.
        home
        mViewPager=(ViewPager) findViewById(R.id.id_viewpager); //获取 ViewPager
        控件
        //实例化 Fragment 及 Fragment 指示器
        initDatas();
        mViewPager.setAdapter(mAdapter); //为 mViewPager 设置适配器
        //为 mViewPager 设置页面滑动监听器, 让 Activity 去实现监听
        mViewPager.setOnPageChangeListener(this);
    }
```




③ 实时数据显示模块 (InfoFragment)。通过 `mApplication.getWSNRConnect()` 获取 `WSNRConnect` 实例, 设置 ID/KEY 和服务地址, 通过“`mApplication.registerOnWSNDataListener(this)`”注册传感器数据监听 (实现接口 `IONWSNDataListener`), 建立实时连接。

```

@Override
public void onMessageArrive(String mac, String tag, String val) {
    //TODO Auto-generated method stub
    if (mac.equalsIgnoreCase(mMac1)) {
        if (mac.equalsIgnoreCase(mMac1)) {
            if (tag.equals("A0")) {
                //输出电压值
                vramd = Double.parseDouble(val);
                mTvVoltage.setText("当前电压: " + fnum.format(vramd) + "V");
            }
            if (tag.equals("A1")) {
                //输出电流值
                eramd = Double.parseDouble(val);
                mTvEletric.setText("当前电流: " + fnum.format(eramd) + "A");
            }
            if (tag.equals("A2")) {
                //输出功率
                pramd = Double.parseDouble(val);
                mTvPower.setText("当前功率: " + fnum.format(pramd) + "W");
            }
        }
    }
    if (mac.equalsIgnoreCase(mMac2)) {
        Log.i("onMessageArrive", "onMessageArrive mMac4");
        if (tag.equals("A0")) {
            double ramd = Double.parseDouble(val); //获取传感器传上来的值
            mTvTemperature.setText("温度值: " + ramd + "℃");
        }
        if (tag.equals("A1")) {
            double ramd = Double.parseDouble(val); //获取传感器传上来的值
            mTvHumility.setText("湿度值: " + ramd + "%RH");
        }
    }
}
}

```

④ 历史记录查询 (HistoryChartFragment): 实例化 `WSNHistory`, 调用 `WSNHistory` 类的历史数据查询方法来查询指定时间段内的数据并以曲线图或者柱状图的形式显示, 在本任务中, 可以选择查询 24 小时电压、24 小时功率或者年度用电量的历史数据, 部分源码如下所示, 详细过程可参考 2.5 节中历史数据查询的流程解析。

```

if (arg2 == 1) { //电压
    channel = channels[0];
    flag2 = 1;
    new getHistoryDataAsyn(1).execute(new String[0]);
}

```



```

if (arg2 == 2) {                                     //功率
    channel = channels[1];
    flag2 = 1;
    new getHistoryDataAsync(2).execute(new String[0]);
}
if (arg2 == 3) {                                     //用电量
    channel = channels[2];
    flag2 = 360;
    new getHistoryDataAsync(3).execute(new String[0]);
}

```

⑤ 余额查询、充值缴费模块 (ChargeFragment): 实例化 WSNProperty 对象, 将用户余额调用用户数据接口中的 put 方法存储到服务器。调用 get 方法获取到用户余额。

//实例化 WSNProperty 对象

```
mWSNProperty = new WSNProperty(InfoFragment.ID, InfoFragment.KEY);
```

调用 put 和 get 方法写入和读取相应数据。

```

if (i == 1) {
    if (Double.parseDouble(balance) < 0)
    {
        balance = "0.0";
        Toast.makeText(getActivity(), "余额不足, 请充值", Toast.LENGTH_LONG).show();
    }
    mWSNProperty.put("balance", balance);                //写入用户余额
}
if (i == 2) {
    remainingResult = mWSNProperty.get("balance");        //读取用户余额
}

```

3. Web 端应用设计

根据 Web 应用编程接口定义, 无线抄表系统的应用设计主要采用实时数据 API 接口、历史数据 API 接口和用户数据 API 接口, JS 部分代码如下。

```

var aid = '123';
var key = '123';

var channel_t = "00:12:4B:00:02:CB:A8:52_A0";           //温度
var channel_h = "00:12:4B:00:02:CB:A8:52_A1";           //湿度
var channel_v = "00:12:4B:00:04:3F:A5:C2_A0";           //电压
var channel_a = "00:12:4B:00:04:3F:A5:C2_A1";           //电流
var channel_p = "00:12:4B:00:04:3F:A5:C2_A2";           //功率
var channel_j = "00:12:4B:00:04:3F:A5:C2_A3";           //电量

var rtc = new WSNRTCConnect(aid, key);
rtc.onConnect = function(){
    console.log("connect to server");
};
rtc.onConnectLost = function(){
    console.log("disconnect from server");
};
rtc.onmessageArrive = function(mac, dat) {
    console.log(mac, ">>>", dat);
    if (mac == mySensorMac1)

```



```

{
    if (dat[0] == '{' && dat[dat.length-1]=='}') {
        dat = dat.substring(1,dat.length-1);
        var its = dat.split(",");
        for (var i=0; i<its.length; i++) {
            var it = its[i].split("=");
            if (it.length == 2) {
                if(it[0] == "A0") { //温度
                    if (gTempChat)gTempChat.setTempValue(parseFloat(it[1]));
                }
                if(it[0] == "A1") { //湿度
                    if (gHumiChat)gHumiChat.setHumiValue(parseFloat(it[1]));
                }
            }
        }
    }
}

if(mac == mySensorMac2)
{
    if (dat[0] == '{' && dat[dat.length-1]=='}') {
        dat = dat.substring(1,dat.length-1);
        var its = dat.split(",");
        for (var i=0; i<its.length; i++) {
            var it = its[i].split("=");
            if (it.length == 2) {
                if(it[0] == "A0") { //电压
                    var vGetValue = parseFloat(it[1]);
                    if (gVChat)gVChat.setVValue(vGetValue);
                }
                if(it[0] == "A1") { //电流
                    var eGetValue = parseFloat(it[1]);
                    if (gAChat)gAChat.setAValue(eGetValue);
                }
                if(it[0] == "A2") { //功率
                    var pGetValue = parseFloat(it[1]);
                    if (gPChat)gPChat.setPValue(pGetValue);
                }
                if(it[0] == "A3") { //电量
                    var electricityGetValue = parseFloat(it[1]);
                    consume(electricityGetValue);
                }
            }
        }
    }
}

};
var myHisData = new WSNHistory(aid, key);

$(document).ready(function(){
    rtc.connect();

```




//获取 24 小时电压

```
myHisData.queryLast1D(channel_v, function(dat){
    var dp = [];

    if(dat.datapoints.length > 0){
        var lt = "";
        for (var i=0; i<dat.datapoints.length; i++) {
            var t = dat.datapoints[i].at.substring(11, 16);
            var ct = t.split(":");
            if (ct[0] != lt) {
                var p = {};
                p['label'] = t;
                p['value'] = dat.datapoints[i].value;
                p['tooltext'] = t+"{br}" + p['value']+'V';
                dp.push(p);
                lt = ct[0];
            }
        }
    }
}
```

```
function ck(){
    if (g24VChat) {
        g24VChat.set24VValue(dp);

        } else {
            setTimeout(ck, 300);
        }
    }
    ck();
};
```

//24 小时功率

```
myHisData.queryLast1D(channel_p, function(dat){
    var dp = [];
    var maxp = 0;
    if(dat.datapoints.length > 0){
        var lt = "";

        for (var i=0; i<dat.datapoints.length; i++) {
            var t = dat.datapoints[i].at.substring(11, 16);
            var ct = t.split(":");
            if (ct[0] != lt) {
                var p = {};
                p['label'] = t;
                p['value'] = dat.datapoints[i].value;
                p['tooltext'] = t+"{br}" + p['value']+'W';
                dp.push(p);
                lt = ct[0];
            }
        }
        if (dat.datapoints[i].value > maxp) maxp = dat.datapoints[i].value;
```



```

    }
}

function ck(){
    if (g24PChat) {
        g24PChat.set24PValue(dp);
    } else {
        setTimeout(ck, 300);
    }
}
ck();
//更新当天最大功率
$("#cd_p").text(maxp + " W");
});

//获取最近一年的用电量
myHisData.queryLast1Y(channel_j, function(dat){
    var dp = [];
    if(dat.datapoints.length > 0){
        var lt = dat.datapoints[0].at.substring(0, 7);
        var lm = lt.split('-')[1];
        var lv = dat.datapoints[0].value;
        for (var i=1; i<dat.datapoints.length; i++) {
            var t = dat.datapoints[i].at.substring(0, 7);
            var ct = t.split("-");
            if (ct[1] != lm || i == dat.datapoints.length-1) {
                var p = {};
                p['label'] = lt;
                p['value'] = dat.datapoints[i].value - lv;
                if (p['value']<0) p['value'] = dat.datapoints[i].value;
                p['tooltext'] = t+"{br}"+p['value']+'kWh';
                dp.push(p);
                lm = ct[1];
                lv = dat.datapoints[i].value;
            }
        }
    }
}

function ck(){
    if (glYJChat) {
        glYJChat.set1YJValue(dp);
    } else {
        setTimeout(ck, 300);
    }
}
ck();
});

//获取最近一周的用电量
myHisData.queryLast5D(channel_j, function(dat){
    var v = 0;
    if(dat.datapoints.length > 1){

```



```
        v = dat.datapoints[dat.datapoints.length-1].value -
                                dat.datapoints[0].value;
        if (v < 0) v = dat.datapoints[dat.datapoints.length-1].value;
    }
    $("#5d_j").text(v+" kWh");
});
//获取最近一月的用电量
myHisData.queryLast1M(channel_j, function(dat){
    var v = 0;
    if(dat.datapoints.length >1){
        v = dat.datapoints[dat.datapoints.length-1].value -
                                dat.datapoints[0].value;
        if (v < 0) v = dat.datapoints[dat.datapoints.length-1].value;
    }
    $("#30d_j").text(v+" kWh");
});
var myDate = new Date();
var m = myDate.getMonth()+1;
var day = myDate.getDate();
var start = myDate.getFullYear()+"-"+m+"-"+day+"T00:00:00Z";
//var end = myDate.getFullYear()+"-"+
//            //(myDate.getMonth()+1)+"-"+myDate.getDate()+"T23:59:59Z";
myHisData.queryLast(channel_j, function(dat){
    var v = 0;
    if(dat.datapoints.length >1){
        v = dat.datapoints[dat.datapoints.length-1].value -
                                dat.datapoints[0].value;
        if (v < 0) v = dat.datapoints[dat.datapoints.length-1].value;
    }
    $("#cd_j").text(v+" kWh");
}, "start="+start);
});
</script>
<!--充值窗口-->
<script type="text/javascript">
    function display() {
        document.getElementById("eject-bg").style.display = "block";
        document.getElementById("eject").style.display = "block";
    }
    function hide() {
        document.getElementById("eject-bg").style.display = "none";
        document.getElementById("eject").style.display = "none";
    }
    window.onload = function ejectTop() {
        var x = window.innerHeight;           //浏览器高度
        var y = 340;                           //eject 高度
        var ejectTop = (x-y)/2;
        document.getElementById("eject").style.top = ejectTop + "px";
    }
    function charge() {
```




```

console.log("charge "+$('#ch_money').val());
var m = parseFloat($('#ch_money').val());
if (isNaN(m)) {
    console.log('错误金额');
    return;
}
balance += m;
myProperty.put('balance', ""+balance, function(){
    $('#balance').text(balance+" RMB");
    $('#balance_2').text(balance);
    hide();
});
}
var userInfo = {
    balance:0,
    electricity:0,
    timestamp:0
};

function consume(j) {
    //var ts = new Date().getTime();
    //var delay = parseInt(Math.random()*(10*1000-3*1000+1)+3*1000);//随
    console.log("当前电量", j);
    if (balance > 0 && last_electricity>0) {
        var q = j - last_electricity;
        if (q < 0) q = j;
        var con = q *1.00 //电价
        balance = balance - con;
        myProperty.put('balance', ""+balance, function(){
            $('#balance').text(balance+" RMB");
            $('#balance_2').text(balance);
        });
    }
    last_electricity = j;
}
var balance = 0;
var last_electricity = 0;
var myProperty = new WSNProperty(aid, key);
$(function(){
    myProperty.get("balance", function(dat){
        balance = parseFloat(dat);
        console.log('get balance '+balance);
        balance = isNaN(balance)?0:balance;
        $('#balance').text(balance+" RMB");
        $('#balance_2').text(balance);
    });
});
});

```

机延时

结合无线抄表系统的应用，可以实现对电表数据的实时采集和传输，为电力系统提供准确的数据支持。



4.7.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个温湿度传感器, 1 个可燃气体传感器 (或其他采集类传感器) 按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到 “C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples” 文件夹下。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 SmartMeter 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入无线抄表系统主界面, 在主界面弹出 “连接网关成功” 消息后即表示连接到智云服务中心。

(4) 连接网关成功后会显示当前传感器采集到的实时数据, 如图 4.56 所示。

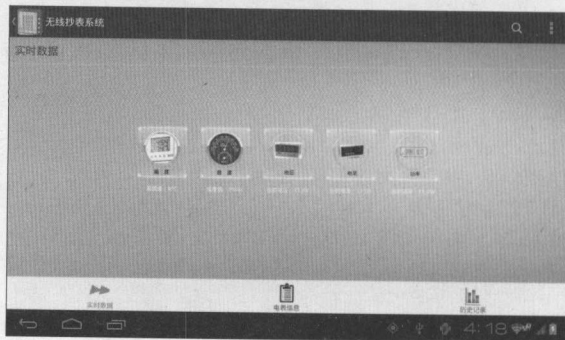


图 4.56 无线抄表系统实时数据显示

(5) 滑动或单击切换至 “电表信息” 页面, 显示当前余额, 还可以单击 “充值” 按钮进行缴费, 如图 4.57 所示。

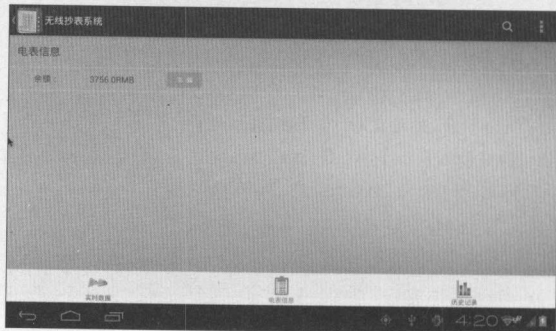


图 4.57 充值缴费模块

(6) 滑动或单击切换至“历史记录”页面，可以选择“24 小时电压”、“24 小时功率”、“月度用电量”来查询历史数据，如图 4.58 所示。

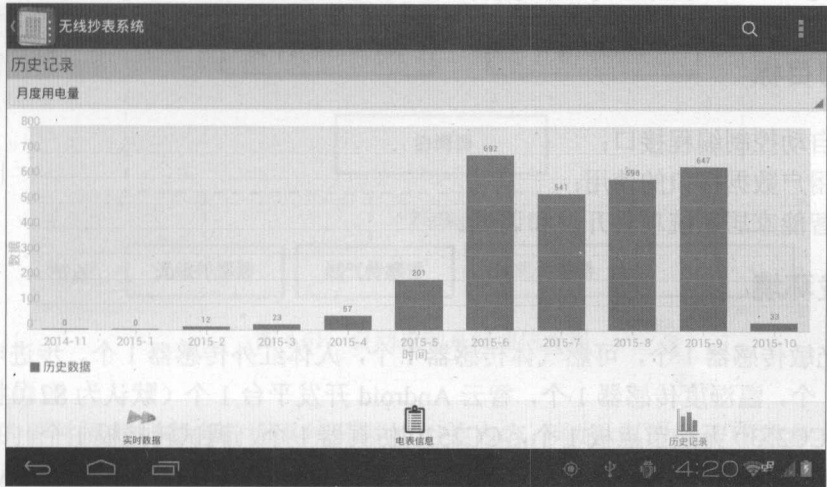


图 4.58 历史数据查询模块

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 电脑接入到互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“SmartMeter-Web\SmartMeter.html”，进入无线抄表系统界面，在左侧栏的上方会显示当前余额，也可以单击“充值”按钮进行余额充值，在左侧栏的下面会实时地显示当前温湿度值，在右侧栏的上方显示当前电流、电压和功率，在右侧栏的下方显示历史消费记录（月度用电量），如图 4.59 所示。

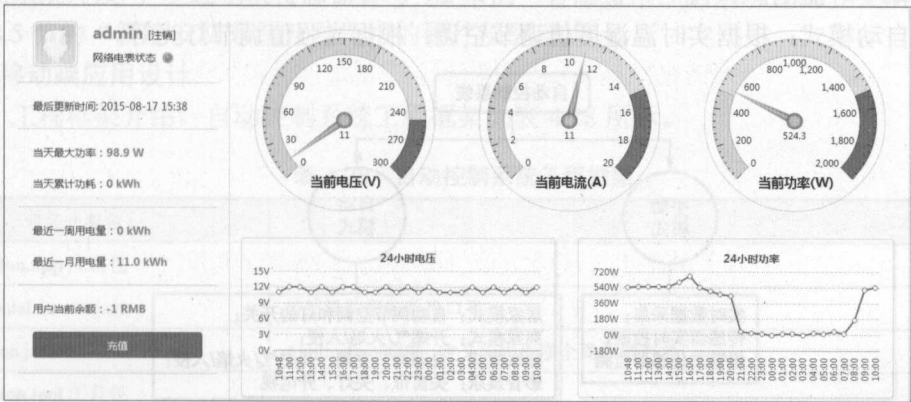


图 4.59 无线抄表网页端设计效果图

4.7.6 总结与拓展

结合无线抄表系统的用户数据接口的使用，可以尝试重新设计智能门禁系统，将用户信息写入到用户数据库。



4.8 任务 26：智能家居自动控制系统开发（案例 14）

4.8.1 学习目标

- 掌握自动控制编程接口；
- 掌握用户数据接口的使用；
- 学会智能家居系统项目开发和调试。

4.8.2 开发环境

硬件：光敏传感器 1 个，可燃气体传感器 1 个，人体红外传感器 1 个，步进电机传感器 1 个，继电器 1 个，温湿度传感器 1 个，智云 Android 开发平台 1 个（默认为 S210 系列 Android 开发平台），CC2530 无线节点板 1 个，CC2530 仿真器 1 个，调试转接板 1 个。

软件：Windows XP/7/8，IAR Embedded Workbench for 8051，Android Developer Tools（Android 集成开发环境）。

4.8.3 原理学习

1. 系统设计目标

自动控制系统中，除了可以手动控制传感器之外，开发者还可以通过调用 ZCloud SDK API 中的自动控制接口实现对传感器的联动控制。例如，在居家模式下，根据温湿度调节空调的开关；根据光强调节客厅灯的开关灯等。通过用户数据接口存储情景模式，当下次进入程序时，会默认选择之前定义的情景模式，如图 4.60 所示。

（1）手动模式：实时数据显示模块。实时显示温湿度、光强值；实时控制灯光等；开启燃气和人体实时监测。

（2）自动模式：根据实时温湿度值调节空调；根据光强值调节灯光等。

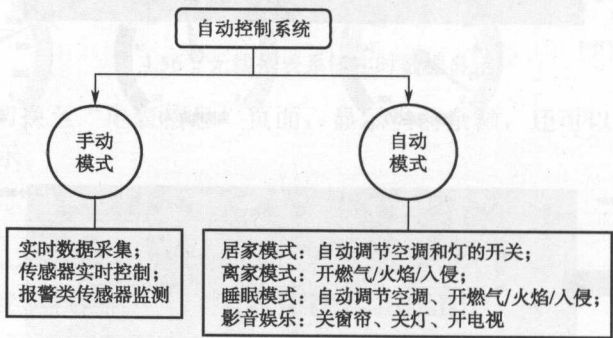


图 4.60 自动控制系统功能模块

2. 业务流程分析

自动控制系统通信流程图如图 4.61 所示。

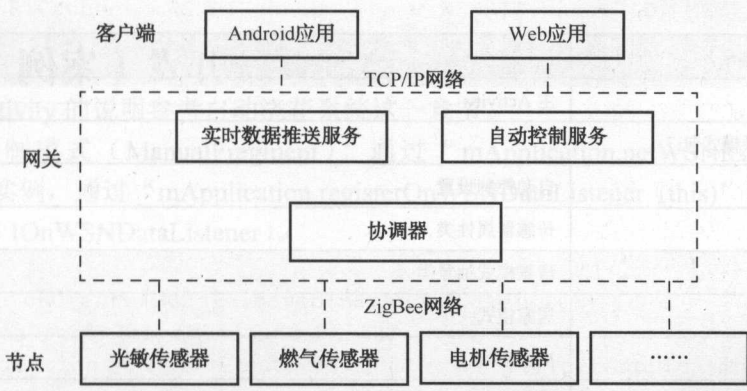


图 4.61 自动控制系统通信流程

3. 硬件原理

传感器的硬件原理可参考前面系统开发的相关内容,例如,温湿度传感器参考 3.1 节的“远程温湿度计系统开发”、继电器参考 3.2 节的“智能灯光控制系统开发”、可燃气体传感器参考 3.3 节中的“厨房燃气检测系统开发”、光敏传感器参考 3.4 节的“农作物光强检测系统开发”、步进电机传感器参考 4.2 节的“智慧窗帘控制系统开发”,人体红外传感器参考 4.5 节的“智能安防系统开发”中的硬件原理。

4.8.4 开发内容

1. 硬件层驱动设计

传感器的硬件层驱动设计可参考前面系统开发的相关内容,例如,例如,温湿度传感器参考 3.1 节的“远程温湿度计系统开发”、继电器参考 3.2 节的“智能灯光控制系统开发”、可燃气体传感器参考 3.3 节中的“厨房燃气检测系统开发”、光敏传感器参考 3.4 节的“农作物光强检测系统开发”、步进电机传感器参考 4.2 节的“智慧窗帘控制系统开发”,人体红外传感器参考 4.5 节的“智能安防系统开发”中的硬件层驱动设计。

2. 移动端应用设计

(1) 工程框架介绍。自动控制系统工程框架如表 4.28 所示。

表 4.28 自动控制系统工程框架

包名(类名)	说 明
com.zonesion.app 应用包	
IONWSNDataListener.java	传感器数据监听接口类
ZApplication.java	Application 对象,定义应用程序全局单例对象
com.zonesion.tool 工具包	
ChangeColorIconWithTextView	自定义 View 的实现
com.zonesion.ui 子模块包	
AutoFragment.java	自动控制模块(情景模式选择)
ManualFragment.java	手动控制模式
com.zonesion.activity activity 包	



续表

包名（类名）	说 明
MainActivity.java	主 Activity
com.zonesion.sh 情景模式包	
ACManage.java	自动控制设置
SensorCfg.java	传感器属性类
Situation.java	情景模式抽象类
HomeSituation.java	居家模式
OutsideSituation.java	离家模式
SleepSituation.java	睡眠模式
VideoSituation.java	影音娱乐模式

(2) 程序业务流程分析。自动控制系统调用的是自动控制服务、实时数据服务和用户数据服务，程序的实现流程如图 4.62 所示。

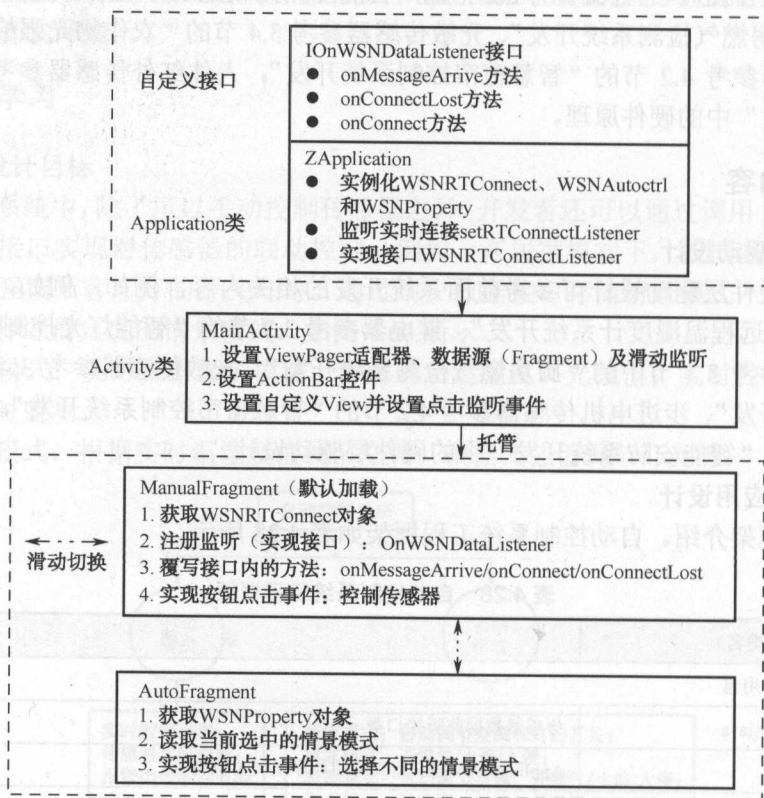


图 4.62 自动控制系统程序实现流程

(3) 程序代码剖析。

① ZApplication 框架说明参考 3.2 节智能灯光控制系统，只是增加了实例化自动控制和用户数据类对象。



```
public WSNRTConnect mWSNRTConnect = new WSNRTConnect(ID, KEY);
public WSNProperty mWSNProperty = new WSNProperty(ID, KEY);
public WSNAutoctrl mWSNAutoctrl = new WSNAutoctrl(ID, KEY);
```

② MainActivity 的说明参考自动浇花系统这一章节。

③ 手动控制模式 (ManualFragment)。通过 “mApplication.getWSNRConnect()” 获取 WSNRConnect 实例, 通过 “mApplication.registerOnWSNDataListener (this)” 注册传感器数据监听 (实现接口 IOnWSNDataListener)。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mZApplication = (ZApplication) getActivity().getApplication();
    mWSNRTConnect = mZApplication.mWSNRTConnect;
    mZApplication.registerOnWSNDataListener(this);
}
```

覆写接口的方法: 在 onConnect()方法中发送查询所有传感器实时数据或者状态的命令, 即在连接成功后立即查询而不是一直等待底层主动上传数据。

```
@Override
public void onConnect() { //连接成功时发送查询命令
    //TODO Auto-generated method stub
    mWSNRTConnect.sendMessage(SensorCfg.saTemperature,
        SensorCfg.cmdQueryTemperature.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saHumidity,
        SensorCfg.cmdQueryHumidity.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saIllumination,
        SensorCfg.cmdQueryIllumination.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saTelevision,
        SensorCfg.cmdQueryTelevision.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saAirConditioner,
        SensorCfg.cmdQueryAirConditioner.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saCurtain,
        SensorCfg.cmdQueryCurtain.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saLivingRoomLamps,
        SensorCfg.cmdQueryLivingRoomLamps.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saSpecificLight,
        SensorCfg.cmdQuerySpecificLight.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saGas,
        SensorCfg.cmdQueryGas.getBytes());
    mWSNRTConnect.sendMessage(SensorCfg.saIntrusion,
        SensorCfg.cmdQueryIntrusion.getBytes());
}
```

在 onMessageArrive()方法中解析获取到的传感器数据, 将实时数据显示在视图中; 根据传感器的状态决定显示的图片背景。以温度实时数据采集 (采集类)、控制电视开关 (控制类) 和燃气监测 (报警类) 为例。

```
@Override
public void onMessageArrive(String mac, String tag, String val) {
    if (mac.equals(SensorCfg.saTemperature)) { //温度实时数据
        if (tag.equals(SensorCfg.chTemperature)) {
            mTVTemperature.setText(val);
        }
    }
}
```



```

    }
}
if (mac.equals(SensorCfg.saTelevision)) { //电视
    if (tag.equals("D1")) {
        flag_tv = Integer.parseInt(val);
        if (flag_tv == 0) {
            mBtTelevision.setBackgroundResource(R.drawable.tvoff);
        } else {
            mBtTelevision.setBackgroundResource(R.drawable.tvon);
        }
    }
}
}
if (mac.equals(SensorCfg.saGas)) { //燃气检测
    if (tag.equals(SensorCfg.chGasStatus)) {
        flag_gas = Integer.parseInt(val);
        if (flag_gas == 0) { //未开启燃气监测
            mBtGas.setBackgroundResource(R.drawable.ranqioff);
        } else { //开启燃气监测
            mBtGas.setBackgroundResource(R.drawable.ranqion);
        }
    }
    if (tag.equals(SensorCfg.chGas)) {
        Double gas = Double.parseDouble(val.trim());
        if (gas >= 10) { //检测到燃气
            System.out.println("检测到燃气超标");
            mBtGas.setBackgroundResource(R.drawable.ranqidanger);
        } else { //未检测到燃气
            System.out.println("检测到燃气未超标");
            mBtGas.setBackgroundResource(R.drawable.ranqisafe);
        }
    }
}
}
}
}

```

单击按钮发送 “{OD1=1, D1=?}{CD1=1, D1=?}” 命令来控制电视的开关。

```

@Override
public void onClick(View v) {
    //TODO Auto-generated method stub
    switch (v.getId()) {
        case R.id.btnTv:
            if (flag_tv == 0) //关闭状态
            {
                mWSNRTConnect.sendMessage(SensorCfg.saTelevision,
                    SensorCfg.cmdOpenTelevision.getBytes());
            } else { //开启状态
                mWSNRTConnect.sendMessage(SensorCfg.saTelevision,
                    SensorCfg.cmdCloseTelevision.getBytes());
            }
            break;
    }
}
}

```



④ 自动控制模式 (AutoFragment)。进入自动控制模式，首先会调用用户数据接口查询已选中的情景模式；开发者选中一种情景模式后，系统会调用用户数据接口存储情景模式至服务器，当下次再进入系统时，会首先读取上次存储的情景模式。

```
public void onCreate(Bundle savedInstanceState) {
    //TODO Auto-generated method stub
    super.onCreate(savedInstanceState);

    //获取 ZApplication 对象
    mApplication = (ZApplication) getActivity().getApplication();
    mWSNProperty = mApplication.mWSNProperty;

    try {
        cur_situation = mWSNProperty.get("_situation");
    } catch (Exception e) {
        cur_situation = "";
        //TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

若当前未选中任何情景模式，系统会弹出消息提示“当前没有选择情景模式”，若已选中情景模式，系统会弹出消息提示开发者选择的情景模式是什么。另外，开发者也可以单击不同的按钮来选择其他情景模式。例如，当选中居家模式时，要实现根据当前温度来调节空调的功能，就需要调用自动控制接口，调用自动控制接口的实现步骤如下。

调用自动控制接口，需要设置触发器、执行器和执行任务。

- 触发器设置：获取 WSNAutoctrl 对象，调用 createTrigger 方法创建触发器，服务器会返回一个 ID，保存服务器返回的触发器的 ID。
- 执行器设置：获取 WSNAutoctrl 对象，调用 createActuator 方法创建执行器，服务器会返回一个 ID，保存服务器返回的执行器的 ID。
- 执行任务设置：获取 WSNAutoctrl 对象，调用 createJob 方法创建执行策略，在创建执行任务时需要触发器和执行器的 ID 作为参数。

```
/*创建温度大于 30 度打开空调任务 */
private void _t_ge_30_o_ac() throws Exception {
    boolean update_a1 = true, update_t1 = true;
    int tid1 = -1, aid1 = -1;
    String jid1 = mWSNProperty.get("jid_t_ge_30_o_ac");
    //String jid2 = mWSNProperty.get("jid_t_le_25_c_ac");
    if (!jid1.equals("")) {
        jid_t_ge_30_o_ac = Integer.parseInt(jid1);
        String sjob = mWSNAutoctrl.getJob(jid1);
        JSONArray a = new JSONArray(sjob);
        if (a.length() == 1) {
            JSONObject job = a.getJSONObject(0);
            JSONObject jobparam = job.getJSONObject("param");
            JSONArray tids = jobparam.getJSONArray("tids");
            if (tids.length() == 1) {
                int tid = tids.getInt(0);
                String stri = mWSNAutoctrl.getTrigger("" + tid);
            }
        }
    }
}
```




```
JSONArray t = new JSONArray(stri);
if (t.length() == 1) {
    JSONObject tri = t.getJSONObject(0);
    JSONObject triparam = tri.getJSONObject("param");
    String tsa = triparam.getString("mac");
    String tch = triparam.getString("ch");
    String top = triparam.getString("op");
    double tva = triparam.getDouble("value");
    if (SensorCfg.saTemperature.equals(tsa) && top.equals(">")
        && (tva >= 29.999999 && tva <= 30.000001)
        && SensorCfg.chTemperature.equals(tch)) {
        update_t1 = false;
        tid1 = tid;
    }
}

JSONArray aids = jobparam.getJSONArray("aids");
if (aids.length() == 1) {
    int aid = aids.getInt(0);
    String sact = mWSNAutoctrl.getActuator("" + aid);
    JSONArray aa = new JSONArray(sact);
    if (aa.length() == 1) {
        JSONObject act = aa.getJSONObject(0);
        JSONObject actparam = act.getJSONObject("param");
        String asa = actparam.getString("mac");
        String acmd = actparam.getString("data");
        if (SensorCfg.saAirConditioner.equals(asa)) {
            if (SensorCfg.cmdOpenAirConditioner.equals(acmd)) {
                update_a1 = false;
                aid1 = aid;
            }
        }
    }
}

if (update_t1) {
    JSONObject po = new JSONObject();
    po.put("mac", SensorCfg.saTemperature);
    po.put("ch", SensorCfg.chTemperature);
    po.put("op", ">");
    po.put("value", 30);
    po.put("once", true);
    String stri = mWSNAutoctrl.createTrigger("_t_ge_30", "sensor", po);
    JSONObject tri = new JSONObject(stri);
    tid1 = tri.getInt("id");
}

if (update_a1) {
```



```

JSONObject po = new JSONObject();
po.put("mac", SensorCfg.saAirConditioner);
po.put("data", SensorCfg.cmdOpenAirConditioner);
String sact = mWSNAutoctrl.createActuator("_o_ac", "sensor", po);
JSONObject act = new JSONObject(sact);
aid1 = act.getInt("id");
}
if (update_t1 || update_a1) { //update_job
    JSONObject po = new JSONObject();
    JSONArray tids = new JSONArray();
    tids.put(tid1);
    JSONArray aids = new JSONArray();
    aids.put(aid1);
    po.put("tids", tids);
    po.put("aids", aids);
    String sjob = mWSNAutoctrl.createJob("_t_ge_30_o_ac", false, po);
    JSONObject job = new JSONObject(sjob);
    jid_t_ge_30_o_ac = job.getInt("id");
    mWSNProperty.put("jid_t_ge_30_o_ac", "" + jid_t_ge_30_o_ac);
}
}

```

3. Web 端应用设计

根据 Web 应用编程接口定义, 自动控制系统的的核心应用设计主要采用自动控制 API 接口、实时数据 API 接口和用户数据 API 接口, JS 部分代码 (主要是情景模式实现部分) 如下。

```

/*情景模式*/
function homeSituation(o) { //居家模式
    if (o) { //open
        mWSNRTConnect.sendMessage(saCurtain, cmdOpenCurtain);
        mWSNRTConnect.sendMessage(saIllumination, cmdCloseIntrusion);
        mWSNProperty.get("jid_t_ge_30_o_ac", function(jid){
            mWSNAutoctrl.setJob(jid, true);
        });
    } else {
        mWSNProperty.get("jid_t_ge_30_o_ac", function(jid){
            mWSNAutoctrl.setJob(jid, false);
        });
    }
}

function outsideSituation(o) { //外出模式
    if (o) {
        //1、关闭窗帘 2、关闭空调 3、关闭电视 4、关闭客厅灯 5、关闭特效灯 6、开启入侵检测
        mWSNRTConnect.sendMessage(saCurtain, cmdCloseCurtain);
        mWSNRTConnect.sendMessage(saAirConditioner, cmdCloseAirConditioner);
        mWSNRTConnect.sendMessage(saTelevision, cmdCloseTelevision);
        mWSNRTConnect.sendMessage(saLivingRoomLamps, cmdCloseLivingRoomLamps);
        mWSNRTConnect.sendMessage(saSpecificLight, cmdCloseSpecificLight);
        mWSNRTConnect.sendMessage(saIntrusion, cmdOpenIntrusion);
    } else {
        //1、关闭入侵监测
    }
}

```



```
mWSNRTConnect.sendMessage(saIntrusion,cmdCloseIntrusion);
    }
}
function sleepSituation(o) { //睡眠模式
    if (o) {
        //1、关闭窗帘 2、关闭客厅灯 3、关闭特效灯 4、关闭电视 5、开启入侵检测
        mWSNRTConnect.sendMessage(saCurtain,cmdCloseCurtain);
        mWSNRTConnect.sendMessage(saLivingRoomLamps,cmdCloseLivingRoomLamps);
        mWSNRTConnect.sendMessage(saSpecificLight,cmdCloseSpecificLight);
        mWSNRTConnect.sendMessage(saTelevision,cmdCloseTelevision);
        mWSNRTConnect.sendMessage(saIntrusion,cmdOpenIntrusion);
    } else {
        //1、关闭入侵监测
        mWSNRTConnect.sendMessage(saIntrusion,cmdCloseIntrusion);
    }
}
function videoSituation(o) { //影音娱乐
    if (o) {
        //1、开启特效灯 2、开启电视
        mWSNRTConnect.sendMessage(saSpecificLight,cmdOpenSpecificLight);
        mWSNRTConnect.sendMessage(saTelevision,cmdOpenTelevision);
    } else {
        //1、关闭特效灯 2、关闭电视
        mWSNRTConnect.sendMessage(saSpecificLight,cmdCloseSpecificLight);
        mWSNRTConnect.sendMessage(saTelevision,cmdCloseTelevision);
    }
}
```

4.8.5 开发步骤

1. 搭建智云硬件环境

(1) 准备一台 S210 系列 Android 开发平台, 1 个光敏传感器, 1 个可燃气体传感器, 1 个直流电机传感器, 1 个人体红外报警器, 1 个继电器传感器, 1 个温湿度传感器。按照任务 3 的方法设置节点板跳线为模式一。

(2) 打开传感器驱动工程: 将本任务 SensorHalExamples 下所有文件夹复制到“C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples”文件夹下。

(3) 分别打开协调器和传感器工程, 编译代码。

(4) 使用 Flash Programmer 工具把程序分别下载到对应的传感器节点板和协调器节点板中, 同时读取传感器节点板的 IEEE 地址。

(5) 参考 2.1 节内容部署硬件, 组成智云无线传感网络, 并将数据接入到智云服务中心。

2. Android 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址及智云 ID/KEY。

(2) 编译 AutoControl 工程, 并安装应用程序到 Android 开发平台或 Android 终端内。

(3) 设置 Android 终端设备接入到互联网或者与智云 Android 开发平台设备在同一个局域网内。进入自动控制系统主界面, 如图 4.63 所示, 系统默认加载 ManualFragment, 显示当前温湿度值和光强值。

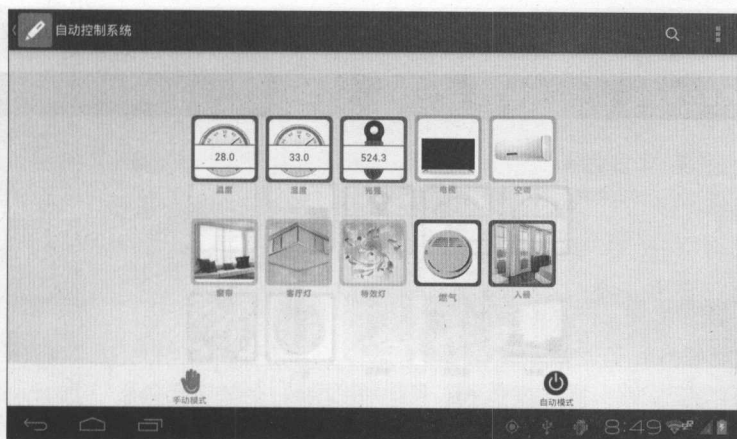


图 4.63 实时显示温湿度/光强值

也可单击按钮控制窗帘、电视等的开关，如图 4.64 所示。

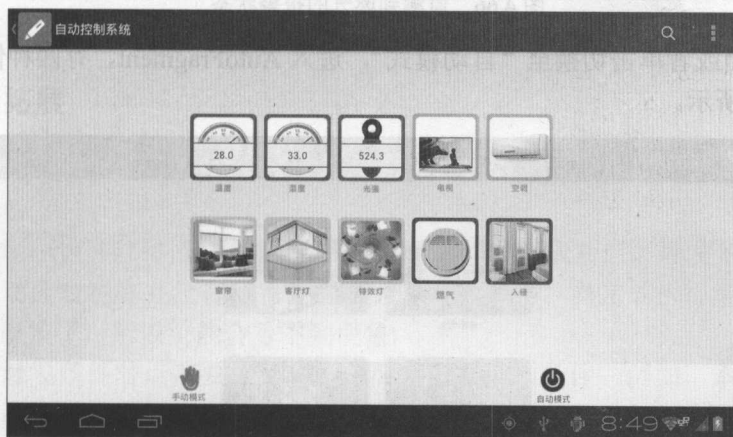


图 4.64 控制电视/空调开关

另外可以单击燃气或入侵按钮来开启燃气监测和人体入侵检测的开关状态，如图 4.65 所示。



图 4.65 开启燃气/人体监测



当检测到燃气浓度超标时，图片会动画闪烁，如图 4.66 所示。

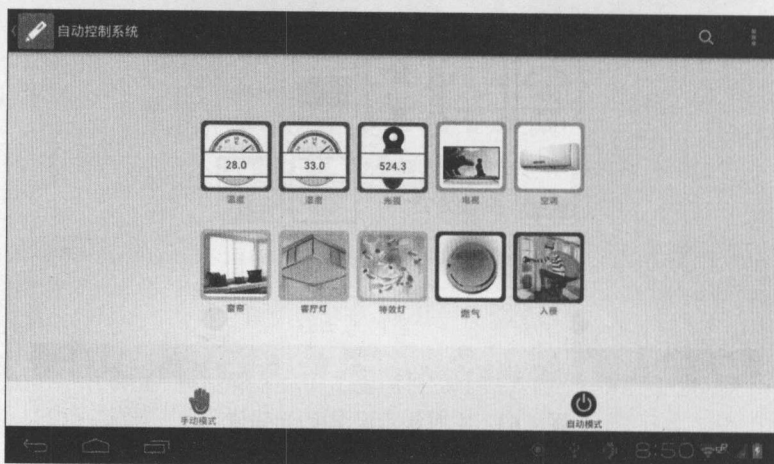


图 4.66 监测到燃气的报警状态

(4) 滑动页面或者单击切换至“自动模式”，进入 AutoFragment，有四种情景模式供用户选择，如图 4.67 所示。

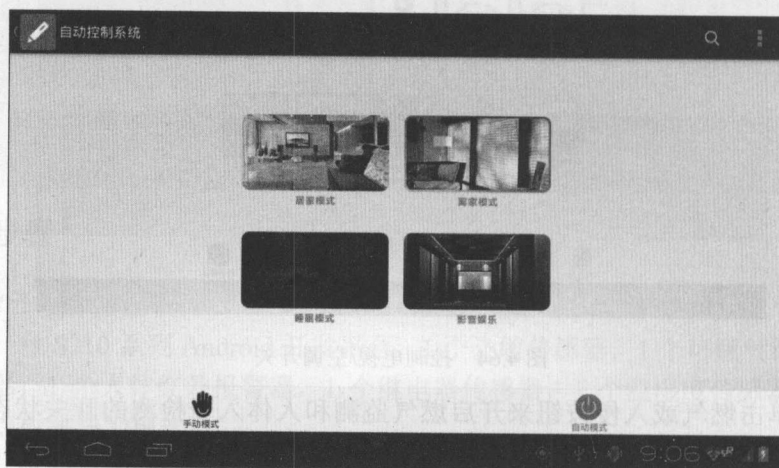


图 4.67 自动控制模块

3. Web 应用程序开发

(1) 根据实际硬件平台修改代码中传感器节点的 IEEE 地址、智云服务器地址（若在局域网内使用，则设置为智云 Android 开发平台的 IP）和智云 ID/KEY。

(2) 将计算机接入互联网，或与智云 Android 开发平台设备在同一个局域网内。用谷歌浏览器（或支持 HTML5 技术的 IE10 以上版本浏览器）运行 Web 工程“AutoControl-Web\AutoControl.html”，进入自动控制系统界面，如下图所示。界面上方显示的是实时数据（温湿度、光强值），界面左侧是四种供用户选择的情景模式（采用自动控制接口实现），界面中间是自动控制系统中所有传感器，可以单击开关对其进行控制，如图 4.68 所示。



图 4.68 自动控制系统 Web 实现

4.8.6 总结与拓展

本任务实现自动控制采用了自动控制 API 接口,调用自动控制 API 接口需要配置触发器、执行器和执行任务。

调用自动控制 API 接口时需要注意的是,当保存触发器成功后,触发条件一旦满足,触发器就会触发,而此时也许并没有设置好执行器和执行策略,当完成执行策略的保存后,实际上触发器已经触发了,所以可能会看不到现象。

因此在设置触发条件时,需要根据实际情况选择可行的条件。例如,本任务中,温湿度传感器采集到的温度值一般是 28° , 设置触发条件为:温度高于 25° 时则开启空调,一旦温度由低于 25° 转为高于 25° 时,触发器就会触发,而如果温度一直处于高于 25° 的状态,触发器并不会一直被触发。

采用调用自动控制接口的好处在于:在服务器端定义好自动控制接口后,当同一个网关内连接多个客户端(Android 端)时,设置的自动控制执行任务对多个客户端都是有效的,因为所有的客户端请求的都是同一个服务器。相比于在 Android 端实现自动控制功能的设计,可以简化客户端的开发。

常见硬件及问题

A.1 Android 智云 Android 开发平台的使用

支持各种 Android 开发平台和 Android 终端设备，通过内置应用程序“智云服务配置工具”进行配置，通过集成的无线 SINK 节点（协调器）可支持 ZigBee、Wi-Fi、蓝牙 BLE4.0、RF433M、IPv6 等多种无线网络接入，支持 Wi-Fi、以太网、3G 等网络接入到移动网/电信网。

Android 智云 Android 开发平台配置如下。

- (1) 将网关通过 3G/Wi-Fi/以太网任意一种方式接入到互联网（若仅在局域网内使用，可不用连接到互联网），在智云 Android 开发平台的 Android 系统运行程序——智云服务配置工具。
- (2) 在用户账号、用户密钥栏输入正确的智云 ID/KEY，也可单击“扫一扫二维码”按钮，用摄像头扫描购买的智云 ID/KEY 所提供的二维码图片，自动填写 ID/KEY（若数据仅在局域网使用，可任意填写）。
- (3) 服务地址为 zhiyun360.com，若使用本地搭建的智云数据中心服务，则填写正确的本地服务地址。
- (4) 单击“开启远程服务”按钮，成功连接智云服务后则支持数据传输到智云数据中心；单击“开启本地服务”按钮，成功连接后智云服务将向本地进行数据推送，如图 A.1 所示。

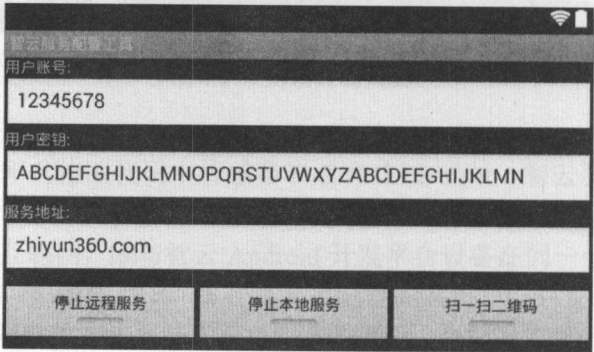


图 A.1

根据实际需要，需要对接入的节点参数进行设置。

- (1) 在智云服务配置工具主界面，按下“MENU”按键，弹出“无线接入设置”菜单，单



点击进入菜单，弹出节点参数配置节点，支持四种无线网络的接入，可根据接入的网络对选项进行勾选，如图 A.2 所示。

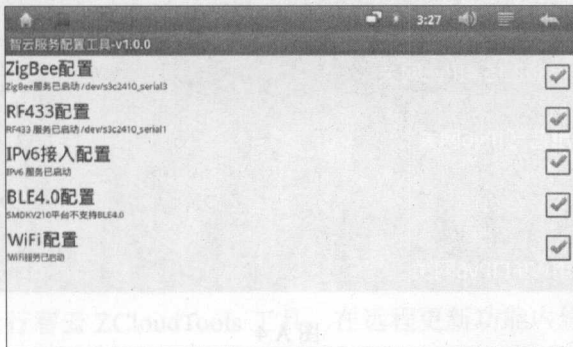


图 A.2

(2) 直接单击各个子项，可进入到各个无线节点参数配置界面。

ZigBee 配置：弹出的菜单选择协调器所接入的串口（默认该服务会自动判别智云 Android 开发平台的串口设置）。

RF433 配置：弹出的菜单选择协调器所接入的串口。

IPv6 接入配置：IPv6 智云接入服务主要是针对于 IPv6 多网融合传感网络节点接入，可支持运行 Contiki IPv6 协议的 802.15.4、蓝牙、Wi-Fi 三种无线节点接入。智云工具弹出的界面，可分别对 802.15.4、蓝牙、Wi-Fi 三种运行 IPv6 协议栈的节点进行配置（可参考 IPv6 多网融合开发平台的手册进行设置），如图 A.3 所示。

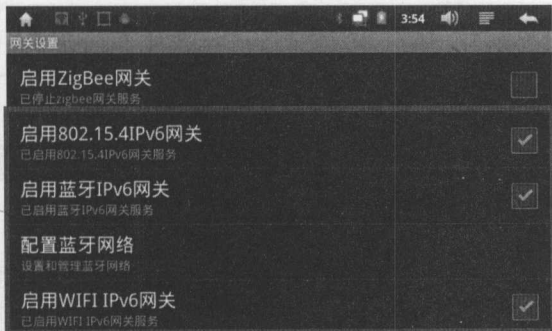


图 A.3

说明：对于 ZigBee 网络服务，智云 Android 开发平台默认兼容早期 ZigBee 演示程序，再使用智云服务时，需要确保串口未被占用，在“无线接入设置”的界面，按下“MENU”按键，弹出“其他设置”菜单，单击进入菜单，在弹出的界面将“启用 ZigBee 网关”选项关闭，如图 A.4 所示。

BLE4.0 配置：BLE4.0 智云接入服务主要是针对于 CC2540 无线传感网的接入。蓝牙 BLE 服务要求使用 S4418 系列开发平台作为网关使用，或者安装有智云服务的 Android 终端平台/手机使用。使用前要求先将系统的无线和网络设置中的蓝牙打开。

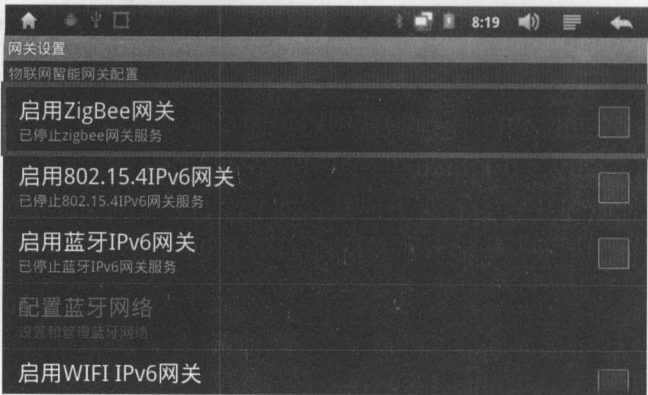


图 A.4

单击 BLE4.0 配置子项进入到设置界面，软件会自动搜索周边的 BLE 设备，单击搜索到的设备，将会允许设备入网，在入网设备列表可以看到已经入网的设备（单击可取消入网）。返回无线接入设置勾选“BLE4.0 服务”可开启该服务，如图 A.5 所示。

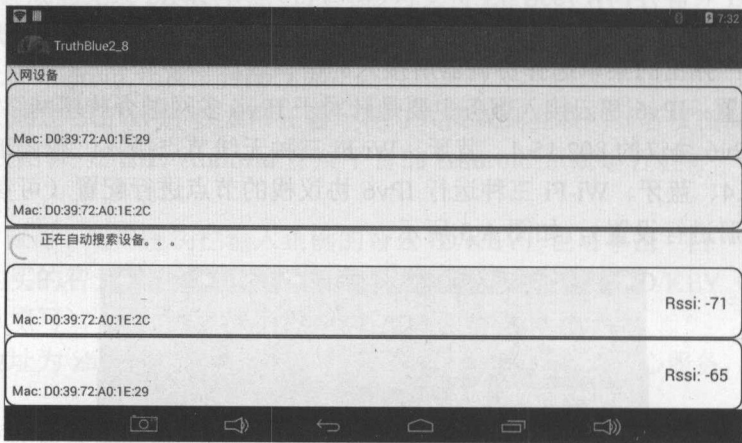


图 A.5

Wi-Fi 配置：Wi-Fi 智云接入服务主要是针对于 CC3200 无线传感网的接入，勾选 Wi-Fi 服务可开启该服务。Wi-Fi 节点可设置连接到与网关所在的无线局域网内，也可单击 Wi-Fi 配置子项进入网关 Wi-Fi 热点配置程序设置为 AP 模式使得 Wi-Fi 节点接入。

A.2 无线节点镜像固化

ZigBee 无线节点/协调器镜像固化。

- (1) 安装 TI CC2530 程序下载工具——SmartRF Flash Programmer（本书配套资料“DISK\05-常用工具\ZigBee\Setup_SmartRFProgr_1.12.4.exe”）
- (2) 将 CC2530 仿真器通过调试转接板连接到节点的调试接口槽，另一端通过 USB 线缆接入到电脑（驱动默认位置（默认位置为“C:\Program Files (x86)\Texas Instruments\SmartRF Tools\Drivers\Cebal”））。



(3) 运行 SmartRF Flash Programmer 程序,“program”下拉菜单选择“Program CCxxxx SoC or MSP430”,此时“System-on-Chip”选项卡可以看到已经识别了仿真器为 SmartRF04EB 和节点芯片类型为 CC2530,如果没有看到仿真器,则按一下仿真器的复位按钮或重新插拔仿真器的 USB 线缆,在 Flash image 选项选择要固化的镜像文件,单击“Perform actions”按钮开始固化镜像文件,固化成功后,会提示“Erase, program and verify OK”则表示镜像固化成功。

A.3 无线节点修改网络信息

ZigBee 无线节点/协调器节点的网络信息修改方法有两种:一是通过修改源码重新编译成新的镜像文件,二是运行智云 ZCloudTools 工具,在远程更新功能内修改。

1. 方法一:通过源码修改

(1) 安装 TI ZStack 源码包,见本书配套资料“DISK\03-ZigBee 安装资料\ZStack\ZStack-CC2530-2.4.0-1.4.0.exe”,默认安装路径为 C 盘根目录。

(2) 修改配置文件“C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Tools\CC2530DB\fwConfig.cfg”。

根据需要 PAN ID 修改为四位数。

```
-DZDAPP_CONFIG_PAN_ID=0x2100
```

也可以修改 CHANNEL 信道(选择其中一个信道,注释掉其他)。

```
// -DDEFAULT_CHANLIST=0x04000000 //26 - 0x1A
// -DDEFAULT_CHANLIST=0x02000000 //25 - 0x19
// -DDEFAULT_CHANLIST=0x01000000 //24 - 0x18
// -DDEFAULT_CHANLIST=0x00800000 //23 - 0x17
// -DDEFAULT_CHANLIST=0x00400000 //22 - 0x16
// -DDEFAULT_CHANLIST=0x00200000 //21 - 0x15
// -DDEFAULT_CHANLIST=0x00100000 //20 - 0x14
// -DDEFAULT_CHANLIST=0x00080000 //19 - 0x13
// -DDEFAULT_CHANLIST=0x00040000 //18 - 0x12
// -DDEFAULT_CHANLIST=0x00020000 //17 - 0x11
// -DDEFAULT_CHANLIST=0x00010000 //16 - 0x10
// -DDEFAULT_CHANLIST=0x00008000 //15 - 0x0F
// -DDEFAULT_CHANLIST=0x00004000 //14 - 0x0E
// -DDEFAULT_CHANLIST=0x00002000 //13 - 0x0D
// -DDEFAULT_CHANLIST=0x00001000 //12 - 0x0C
-DDEFAULT_CHANLIST=0x00000800 //11 - 0x0B
```

(3) 将对应的节点工程源码(本书配套资料“DISK\04-开发例程\Chapter02 智云物联开发基础\任务 05-智云硬件驱动开发\”目录下所有文件夹资料)复制到默认工程目录下“C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples\”,打开 eww 工程文件,单击 IAR 环境菜单栏 Project→Rebuild All,重新编译源码,即可工程目录下获取到编译好的 hex 格式镜像。

2. 方法一:ZCloudTools 工具修改

运行 ZCloudTools 程序,进入到远程更新功能模块,左侧节点列表列出了组网成功的节点设备(PID=8212 CH=11 <节点 MAC 地址>),其中 PID 表示节点设备组网的 PANID,CH 表示其组网的 CHANNEL。依次单击复选框,选择所要更新的节点设备,输入 PANID 和 CHANNEL



号，单击“一键更新”按钮，执行更新，如图 A.6 所示。



图 A.6

注意：此处 PANID 的值为十进制，而底层代码定义的 PANID 的值为十六进制，需要自行转换。例如，8200(十进制) = 0x2008(十六进制)，通过“{PANID=8200}”命令将节点的 PANID 修改为 0x2008。

A.4 无线节点读取 IEEE 地址

ZigBee 无线节点/协调器节点的读取 IEEE 地址的方法如下。

(1) 安装 TI CC2530 程序下载工具——SmartRF Flash Programmer（见本书配套资料“DISK\05-常用工具\ZigBee\Setup_SmartRFProgr_1.12.4.exe”）。

(2) 将 CC2530 仿真器通过调试转接板连接到节点的调试接口槽，另一端通过 USB 线缆接入到电脑（默认驱动默认位置为“C:\Program Files (x86)\Texas Instruments\SmartRF Tools\Drivers\Cebal”）。

(3) 运行 SmartRF Flash Programmer 程序，在“Program”下拉菜单选择“Program CCxxxx SoC or MSP430”，此时“System-on-Chip”选项卡可以看到已经识别了仿真器为“SmartRF04EB”和节点芯片类型为“CC2530”，如果没有看到仿真器，则按一下仿真器的复位按钮或重新插拔仿真器的 USB 线缆，找到并单击“Read IEEE”按钮，在页面的“IEEE 0x”一栏就会显示 CC2530 的 MAC 地址信息，如图 A.7 所示。

说明：默认无线节点采用的是主 IEEE 地址（全球唯一不可修改），但根据实际需求用户可以采用扩展地址，可在上述软件的“Location”中选择“Secondary”进行读取/写入扩展地址，则此时节点启动时将默认使用该地址作为 IEEE 地址。当不使用扩展地址时，选择“Erase”擦除 Flash 即可。

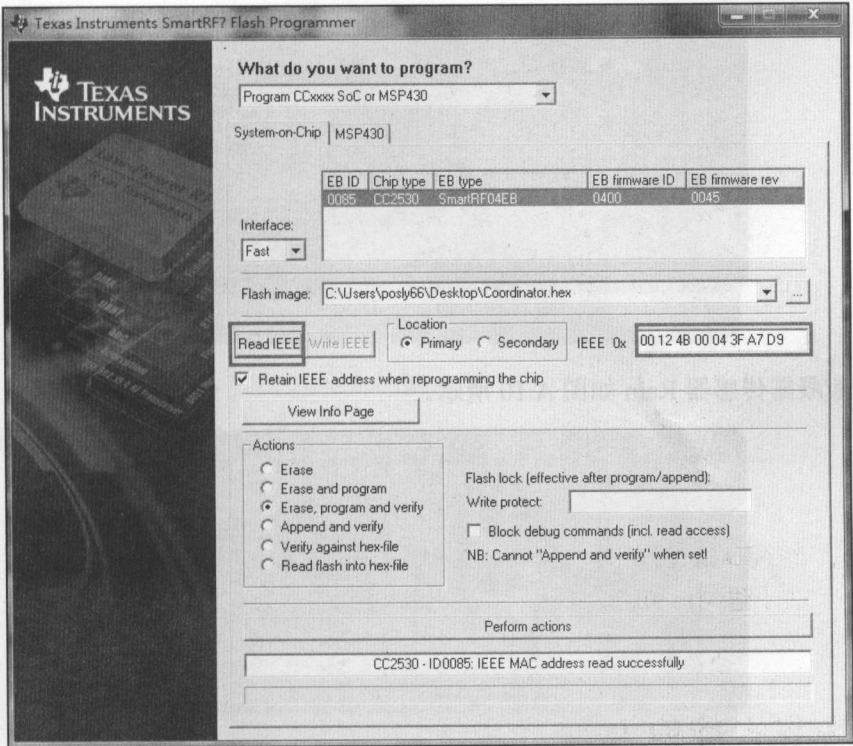


图 A.7

A.5 认识各种传感器

下面是各种传感器模块图片。

(1) 可燃气体 CombustibleGas 如图 A.8 所示。

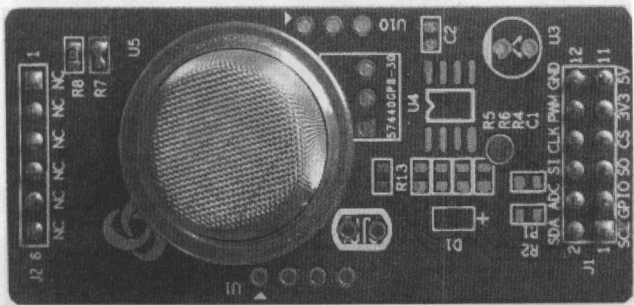


图 A.8

空气质量与可燃气体传感器外形一样，只是型号不同。

(2) 酒精传感器 AlcoholGas 如图 A.9 所示。

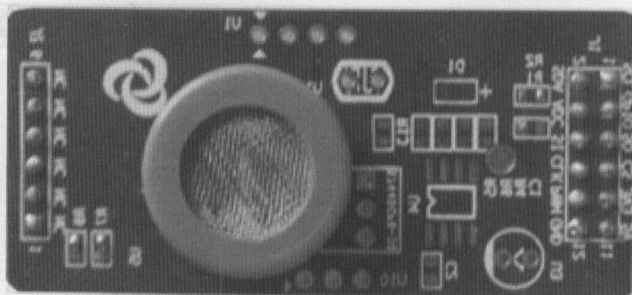


图 A.9

(3) 雨滴/凝露传感器 Rain 如图 A.10 所示。

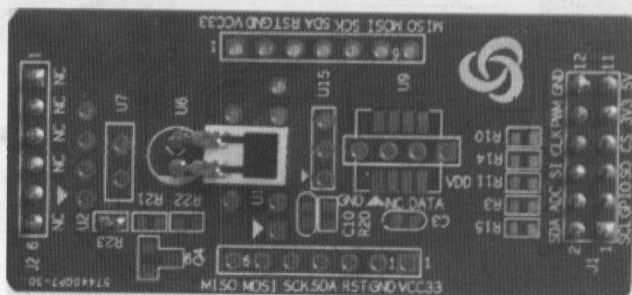


图 A.10

(4) 火焰传感器 Flame 如图 A.11 所示。

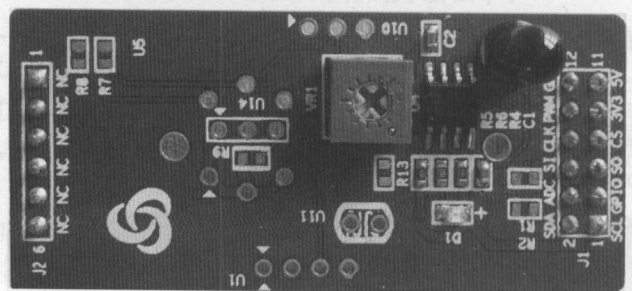


图 A.11

检测火焰前，需要将电位器调节至 LED 灯刚刚灭。

(5) 光敏传感器 Photoresistance 如图 A.12 所示。

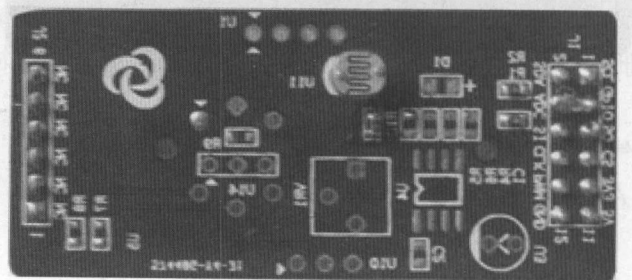


图 A.12

(6) 霍尔传感器 Hall 如图 A.13 所示。

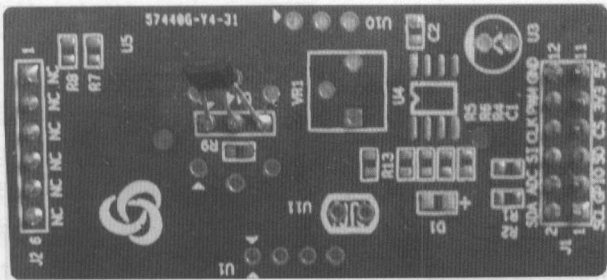


图 A.13

(7) 压力传感器 Pressure 如图 A.14 所示。

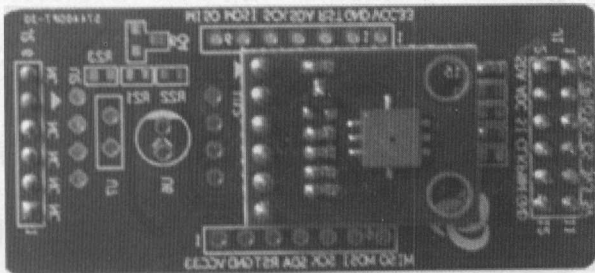


图 A.14

(8) 三轴传感器 Acceleration 如图 A.15 所示。

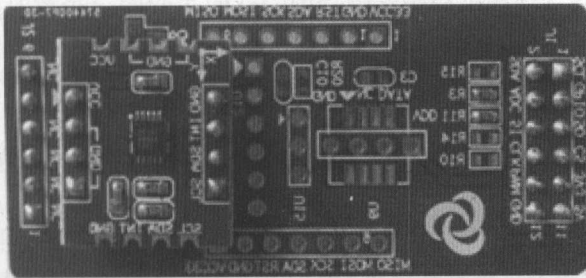
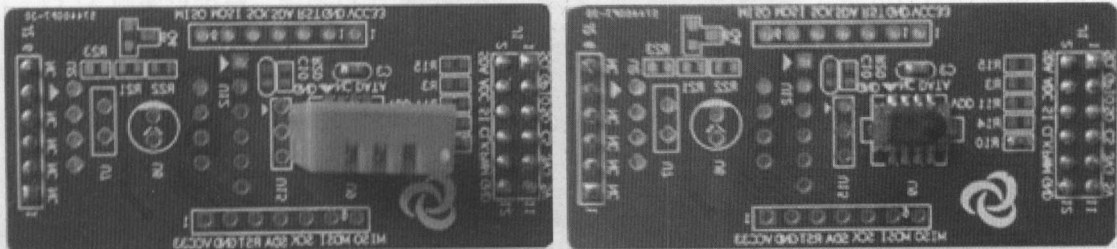


图 A.15

(9) 温湿度传感器 HumiTemp（分两种）如图 A.16 所示。



(a) DHT11

(b) SHT1x

图 A.16



(10) 人体红外 Infrared 如图 A.17 所示。

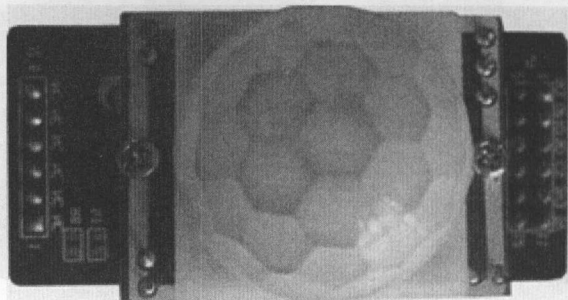


图 A.17

(11) 超声波测距传感器 Ultrasonic 如图 A.18 所示。

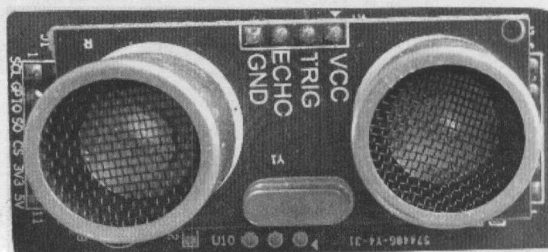


图 A.18

(12) 继电器 Relay 如图 A.19 所示。



图 A.19

(13) RFID 如图 A.20 所示。



图 A.20



A.6 传感器操作说明

(1) 继电器。通过界面上的开关按钮控制继电器的开合，可以看到传感器上对应的 LED 指示灯亮灭，听到继电器吸合的“咔嚓”声音。

(2) 超声波。使用不透光的书本（遮挡面积要大）遮挡模块，可以检测出书本与模块间的距离，当距离低于指定值时会提示报警。

(3) 火焰。调节模块电位器使模块上的 LED 由亮至刚刚灭，用打火机点火（火焰尽量大些），靠近传感器（注意不要烧到传感器），提示检测到火焰，图标变亮，同时网关播放报警声。

(4) 可燃气体。打火机点火，吹灭火焰，靠近传感器，图标全部变亮，同时网关播放报警声。

(5) 空气质量。打火机点火，吹灭火焰，靠近传感器，图标全部变亮，同时网关播放报警声。

(6) 酒精。用盛酒的酒瓶口靠近传感器，图标随着浓度的变大而一个个变亮，全部亮时网关会播放报警声。

(7) 霍尔。当检测到磁铁靠近（注意极性），提示检测到磁场，图标变亮，同时网关播放报警声。

(8) 人体红外。当检测到人体靠近（移动的人体），提示检测到人体，图标变亮，同时网关播放报警声。

(9) 光敏。用手捂住模块，图标全部灭，同时网关播放报警声；用手电筒照射模块，图标全部亮，同时网关播放报警声。

(10) 红外避障。将模块对准障碍物，提示检测到障碍物，图标变亮，同时网关播放报警声。

(11) 流量计数传感器。可以通过对着进水口（值有变化的一端）吹气来实验，同时观察计数值的变化。

(12) 防水温度。用手握住模块，观察温度的变化。

(13) 噪声。调节模块电位器使模块上的 LED 由亮至刚刚灭，在模块附近拍手，提示检测到噪声，图标变亮，同时网关播放报警声。

(14) 温湿度。对着模块哈气，观察温湿度变化。

(15) 雨滴传感器。雨滴传感器采用的 HDS10 元件是正特性开关型元件，对低湿不敏感而仅对高湿敏感，所以当对着传感器哈气时，检测值会迅速变高，图标会显示雨天，停止哈气后必须过一段时间检测值才会降下来。

(16) 三轴。进入界面后显示 X、Y、Z 轴的值，倾斜节点观察值的变化。

(17) 压力。进入界面后显示大气压的值，用手指按压模块，观察值的变化。

(18) 高频 RFID。将卡靠近模块，读取到卡号，添加对应的用户 ID，再次将卡靠近模块，提示卡号及用户 ID。

(19) 振动。轻轻敲击节点，提示检测到振动，图标变亮，同时网关播放报警声。

(20) 触摸。用手指触摸模块，提示检测到触摸，图标变化，同时网关播放报警声。



(21) 步进电机。设置转动角度，单击对应按钮控制电机的转动。

(22) 风扇。单击对应按钮控制风扇的转动，界面显示当前的转速。

(23) 语音识别。进入语音识别传感器界面后，对着传感器的 mic 说命令，传感器即能识别说的命令并将它显示出来，同时传感器的蜂鸣器会嘟一声。支持的命令有台灯开、台灯关、窗帘开、窗帘关。

(24) 语音合成。输入需要合成的文本（不超过 7 个汉字或者字符），单击“合成”按钮，模块会播放输入的文本。

(25) 声光报警。单击“开关”按钮控制声光报警模块。

(26) 红外遥控。

须知：

① 由于 CC2530 内存过小，而此模块所需较大内存，所以将此模块采用底板的 STM32 驱动，CC2530 起数据转发作用。出厂默认固化 STM32 驱动镜像，若驱动镜像损坏，可按照 IPv6 光盘内手册中 STM32 镜像固化方法本书配套资料“\DISK\02-镜像\节点-ZXBee\红外遥控.hex”重新固化到底板 STM32 中。

② 跳线设置：串口跳线 CON9、CON10、CON11、CON12 全部跳到 12 脚。

③ 不同于其他传感器通过 CC2530 核心板驱动，红外遥控传感器接法采用底板 STM32F103 驱动，即将双排端插入 J11 插槽。

操作步骤：

① 节点上电后，D6 闪烁、D4 闪烁后熄灭、D5 常亮；入网成功后，D6 常亮。

② 进入红外遥控传感器界面，选择模式（学习模式或遥控模式）。在学习模式下，可以将红外遥控键码存入按钮对应的键值中；在遥控模式下，可以将存储的红外遥控码发送出去。进入界面后默认选择为遥控模式，如图 A.21 所示。



图 A.21

③ 学习键值。选择学习模式，单击按钮 1~9 任意一个，此时 D4 常亮，D5 熄灭，将遥控器对准红外接收头按下待学习的键码（稍微长按）。若学习成功，D4 闪烁后熄灭，D5 常亮；若学习失败或者学习超时（大约 30 s），D4 直接熄灭，D5 常亮。学习成功后可继续学习其他键值，如图 A.22 所示。

④ 键值学习成功后，选择遥控模式。单击按钮 1~9 任意一个即可将保存的对应键值发送出去，D4 闪烁表明发送成功，如图 A.23 所示。



图 A.22



图 A.23

- (27) 土壤湿度传感器。调节模块上的电位器使模块上的 LED 由亮至刚刚灭，进入土壤湿度传感器界面后，可倒一杯自来水（不能为纯净饮用水），将湿度检测端放入水中约 2/3，可观察图标变亮，提示湿度过高，同时网关播放报警声。
- (28) 低频 RFID。将卡靠近模块，读取到卡号，添加对应的用户 ID，再次将卡靠近模块，提示卡号以及用户 ID。

- (29) 直流电机。设置转动速度等级，单击对应按钮控制电机的转动。
- (30) 紧急按钮。按下按钮，图标变化，提示检测到紧急情况，同时网关播放报警声。

参考文献

- [1] 廖建尚. 物联网平台开发及应用——基于 CC2530 和 ZigBee. 北京: 电子工业出版社, 2016.
- [2] 刘云山. 物联网导论. 北京: 科学出版社, 2010.
- [3] 信息化和工业化深度融合专项行动计划(2013—2018). 工信部信(2013)317号. 工业和信息化部.
- [4] 物联网发展专项行动计划. 发改高技(2013)1718号. 国家发展改革委、工业和信息化部等10个部门.
- [5] 物联网“十二五”发展规划. 工业和信息化部.
- [6] 刘艳来. 物联网技术发展现状及策略分析[J]. 中国集体经济, 2013,(09):154-156.
- [7] 国务院关于积极推进“互联网+”行动的指导意见[J]. 中华人民共和国国务院公报, 2015(20):20-22.
- [8] 李新. 无线传感器网络中节点定位算法的研究[D]. 中国科学技术大学, 2008.
- [9] 李振中. 一种新型的无线传感器网络节点的设计与实现[D]. 北京工业大学, 2014.
- [10] 王洪亮. 基于无线传感器网络的家居安防系统研究[D]. 河北科技大学, 2012.
- [11] 沈寿林. 基于 ZigBee 的无线抄表系统设计与实现[D]. 南京邮电大学, 2016.
- [12] 金海红. 基于 Zigbee 的无线传感器网络节点的设计及其通信的研究[D]. 合肥工业大学, 2007.
- [13] 彭瑜. 低功耗、低成本、高可靠性、低复杂度的无线电通信协议——ZigBee[J]. 自动化仪表, 2005,(05):1-4.
- [14] Alliance Z B. ZigBee Specification[J]. 2007, 1(1).
- [15] Texas Instrument. Z-Stack Compile Options.pdf.
- [16] 樊明如. 基于 ZigBee 的无人值守的酒店门锁系统研究[D]. 安徽理工大学, 2014.
- [17] 陈明燕. 基于 ZigBee 温室环境监测系统的研究[D]. 西安科技大学, 2012.
- [18] Texas Instrument.Z-StackDeveloper's Guide
- [19] 陈海明, 崔莉, 谢开斌. 物联网体系结构与实现方法的比较研究[J]. 计算机学报, 2013,01:168-188.
- [20] 廖建尚. 基于物联网的温室大棚环境监控系统设计方法[J]. 农业工程学报, 2016,11:233-243.
- [21] 邱杰凡, 钱丽萍, 黄亮, 等. 面向物联网重编程的存储优化方法研究[J]. 计算机学报, 2016,39:1-15.
- [22] 周源, 周志雄, 童成彪. 物联网形式的供水管网压力控制管理系统[J]. 计算机工程与应用, 2016, 10:1-6.
- [23] 吴滨, 黄庆展, 毛力, 等. 基于物联网的水产养殖水质监控系统设计[J]. 传感器与微系统, 2016,11:1-4.
- [24] 苗凤娟, 高玉峰, 陶佰睿, 等. 基于物联网与太阳能光伏的智能温室监控系统设计[J]. 科技通报, 2016,09:89-92.



- [25] 臧贺藏, 王言景, 张杰, 等. 基于物联网技术的设施作物环境智能监控系统[J]. 中国农业科技导报, 2016(05).
- [26] 倪明选, 张黔, 谭浩宇, 等. 智慧医疗—从物联网到云计算[J]. 中国科学: 信息科学, 2013,04:515-528.
- [27] 葛文杰, 赵春江. 农业物联网研究与应用现状及发展对策研究[J]. 农业机械学报, 2014,07:222-230+277.
- [28] 孙其博, 刘杰, 黎彝, 等. 物联网: 概念、架构与关键技术研究综述[J]. 北京邮电大学学报, 2010,03:1-9.
- [29] [26] 秦琳琳, 陆林箭, 石春, 等. 基于物联网的温室智能监控系统设计[J]. 农业机械学报, 2015,03:261-267.
- [30] [27] 李正民, 张兴伟, 柳宏川. 基于 CC2530 的温湿度监测系统的设计与实现[J]. 测控技术, 2013,05:25-28, 39.
- [31] 廖建尚. ARM9 和 Linux 的 DS18B20 驱动程序研究[J]. 单片机与嵌入式系统应用, 2013,04:53-56.
- [32] 蔡利婷, 陈平华, 罗彬, 等. 基于 CC2530 的 ZigBee 数据采集系统设计[J]. 计算机技术与发展, 2012,11:197-200.
- [33] 廖建尚. 基于 CC2530 和 ZigBee 的智能农业温湿度采集系统设计[J]. 物联网技术, 2015,08:25-29.
- [34] Texas Instruments. Z-Stack developer's guide[M]. California USA: Texas Instruments, 2015.
- [35] ZigBee Alliance. ZigBee Speciafication[S]. USA: ZigBee Alliance, 2008.
- [36] Texas Instruments. Z-Stack user's guide for smartrf05eb and CC2530. California USA: Texas Instruments, 2011.
- [37] CC253x System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee® Applications User's Guide
- [38] 邓中华. ZigBee 技术在无线温度采集系统中的应用研究[D]. 华南理工大学, 2011.
- [39] 俞建. 基于 ZigBee 无线传感网络的 LED 智能照明控制系统的研究[D]. 浙江工业大学, 2012.
- [40] 彭立. 基于 GSM 和 GPS 的运输车辆状态监管系统的设计[D]. 华南理工大学, 2014.
- [41] 郭秋鉴. 基于网络的摄像头云台控制系统[D]. 西安工业大学, 2011.
- [42] DHT11 Humidity & Temperature Sensor. <http://www.micro4you.com/files/sensor/DHT11.pdf>.
- [43] 高婷. 基于北斗定位的海上落水报警装置设计与研究[D]. 上海海洋大学, 2014.
- [44] 数字温湿度传感器 DHT11.
- [45] 宋景文. 火焰传感器[J]. 自动化仪表, 1991,(05):5-6.
- [46] 刘振照. 基于 OpenGL 的继电器三维可视化仿真系统的研究与开发[D]. 福州大学, 2006.
- [47] 范洪亮. 基于红外传感器的地铁隧道监测系统的设计[D]. 黑龙江大学, 2015.
- [48] 张群强, 赵巧妮. 基于 MQ-2 型传感器火灾报警系统的设计[J]. 价值工程, 2015,(13):96-98.



- [49] 郭坚. 基于 SIM908 的无人机空气质量监测系统设计与研究[D]. 天津大学, 2014.
- [50] MFRC522 datasheet http://www.dzsc.com/datasheet/MFRC522_2417831.html.
- [51] 黄俊祥, 陶维青. 基于 MFRC522 的 RFID 读卡器模块设计[J]. 微型机与应用, 2010,(22):16-18.
- [52] <http://developer.android.com/>.
- [53] Phillips Bill Brian Hardy. Android Programming. the big nerd ranch gui 2013-9.
- [54] Meier Reto.professional Android 4 application development. Wrox. 2012-05

物联网&云平台 高级应用开发



本书涉及嵌入式系统和物联网系统的开发技术，将CC2530接口技术、传感器驱动、ZigBee无线传感网络技术、物联网平台开发技术、Android移动互联网开发技术结合在一起，实现了强大的物联网数据采集、传输和处理，可以开发功能强大的物联网系统，并可适用在多个行业。

采用任务式开发的学习方法，共14个趣味盎然、贴近生活的案例，每个案例均有完整的开发过程，分别是明确的学习目标、清晰的环境开发要求、深入浅出的原理学习、详细的开发内容和完整的开发步骤，最后进行总结与拓展，每个案例均有完整的开发源代码，在此基础上可以快速进行二次开发。

本书配有开发资源包，资料翔实，可加深读者对物联网开的理解及应用。



电子信息出版分社微博
<http://weibo.com/etpublish>



责任编辑：田宏峰
封面设计：汇众智捷



官方微信平台

ISBN 978-7-121-31106-2



9 787121 311062 >

定价：68.00元